Jupiter ACE

FORTH PROGRAMM-HANDBUCH

Steven Vickers

Deutsche Übersetzung

Friedrich Haas & Wolfgang Diez

Es kann keine Gewähr dafür übernommen werden, daß die in diesem Buch verwendeten Angaben, Schaltungen und Programmlistings frei von Schutzrechten Dritter sind. Alle Angaben werden nur für Amateurzwecke gemacht. Alle Daten und Vergleichsangaben sind als unverbindliche Hinweise zu verstehen.

Nachdruck und öffentliche Wiedergabe, besonders die Übersetzung in andere Sprachen, verboten. Programmlistings dürfen weiterhin nicht in irgendeiner Form vervielfältigt und verbreitet werden. Alle Programmlistings sind Copyright der Firma reflecta electronic GmbH. Verboten ist weiterhin die öffentliche Vorführung und Benutzung dieser Programme in Seminaren und Ausstellungen. Irrtum, sowie alle Rechte vorbehalten.

Copyright der englischen Originalausgabe by
Jupiter Cantab Ltd. © 1982

Copyright der deutschen Ausgabe by
reflecta electronic GmbH
Berlichingenstr.9, 8540 Schwabach
© 1983

1. Auflage 1983

Gedruckt in der Bundesrepublik Deutschland - Printed in West-Germany
Imprimé en RFA

Jupiter ACE

FORTH PROGRAMM-HANDBUCH

Steven Vickers

DEUTSCHE ÜBERSETZUNG

Friedrich Haas & Wolfgang Diez

Die Programmiersprache FORTH ist eine Sprache, die sich im Bereich der Mikrocomputer immer mehr durchsetzt. Für viele Anwendungen ist sie hervorragend geeignet. Mit FORTH kann man, einmal damit vertraut, schnelle und effektive Programme bei hoher Speicherplatzausnutzung schreiben.

Dieses Handbuch, welches für den Benutzer des Jupiter ACE geschrieben wurde und jedem Käufer des ACE zur Verfügung gestellt wird, kann auch unabhängig davon eingesetzt werden, um die hochinteressante Sprache FORTH in klarer und gut lesbarer Darstellung kennenzulernen.

Herausgeber

reflecta electronic GmbH
Berlichingenstr. 9
8540 Schwabach

die auch das Alleinvertriebsrecht für den Jupiter ACE
in der Bundesrepublik Deutschland besitzt

INHALTSVERZEICHNIS

Kapitel	Themen	Seite
	Einführung	5
1	**Aufstellung Ihres Jupiter ACE**	6
2	**Eingabe über die Tastatur** Wie Sie die Instruktionen für den ACE eintasten	8
3	**Programme von der Kassette laden** Wenn Sie fertige Kassetten laden wollen	12
4	**Definition neuer Wörter** Wie Sie eigene Programme schreiben können mit : und ;	14
5	**Einfache Arithmetik** Ganzzahl-Arithmetik und der stack	17
6	**Definition neuer arithmetischer Wörter**	23
7	**Ändern von Wort-Definitionen** Wie Sie Fehler korrigieren können, indem Sie LIST, EDIT und REDEFINE benutzen	30
8	**Wörter, die Zahlen bedeuten** Konstante, Variable und Bytes im Speicher	35
9	**Entscheidungen** Wörter können unter verschiedenen Umständen Unterschiedliches bewirken, indem man IF, ELSE und THEN benutzt	39
10	**Wiederholung** Wörter können das Gleiche wiederholen, indem man BEGIN und DO benutzt	46
11	**Töne und Musik** Benutzung des eingebauten Lautsprechers mit BEEP	55
12	**Der Zeichensatz** und wie Sie Ihre eigenen Zeichen definieren können	59
13	**Zeichnen** mit dem Wort PLOT	67
14	**Programme auf Band speichern** Wie Sie Informationen dauerhaft auf Band speichern können, bevor Sie den ACE ausschalten	73
15	**Brüche und Dezimalpunkte** Fließkommaarithmetik	77
16	**Programmbeeinflussung über Tastatur** so daß Sie ein Programm, während es läuft, kontrollieren und verändern können	81

Kapitel	Themen	Seite
17	**Zahlensysteme** Dezimal, binär, oktal, hexadezimal und andere	87
18	**Boolesche Algebra** Die Wörter AND, OR und XOR	92
19	**Erweiterte Arithmetik** Doppellängen-Arithmetik und formatierte Ausgabe	95
20	**Im Wörterbuch** Wie Sie selbst Wörter definieren können CREATE und DEFINER	101
21	**Zeichenketten und Bereiche** was sie bedeuten und wie man sie definiert	107
22	**Mehrere Wörterbücher** Wie man eigenes Vokabular aufbaut	114
23	**Innerhalb der Doppelpunkt-Definition** Wie man den Umwandlungsprozeß kontrolliert und das Wort COMPILER	117
24	**Speicherauslegung** einschließlich einer Liste von Systemvariablen	123
25	**Maschinencode** für den Z8ØA Prozessor-Chip	129
26	**Wir bauen den ACE weiter aus** Wie Sie eigene elektronische Bauelemente an Ihren ACE anschließen können	132

Anhang

A	Der Zeichensatz	137
B	Fehleranzeigen	142
C	Der Jupiter ACE - Zum Nachschlagen	144
D	Schneller Überblick für FORTH-Kenner	155
	Index	156

EINFÜHRUNG

Im Jahr 1950 fertigte das National Physical Laboratory in Großbritannien den 'Pilot ACE' (Automatic Computing Engine), einen der ersten britischen Computer. Dieser erste ACE hatte eine Speicherkapazität von 1,5 KByte und benötigte 32 Mikrosekunden, um die einfachste Operation zu bewältigen.

Mit einer Unmenge von Kabeln und quecksilberbedampften Röhren brauchte er fast den Platz einer mittleren Küche. Heute kann er im wissenschaftlichen Museum von South Kensington besichtigt werden.

Aufbauend auf dem Pilot ACE entwickelte English Electric den DEUCE (Digital Electronic Universal Computing Engine). Innerhalb von 6 Jahren wurden davon ca. 40 Stück verkauft, zu einem Stückpreis von £ 30-40.000

Im Jahr 1982 hat Jupiter Cantab Ltd. seinen eigenen ACE entwickelt.

Er kann schon in der Basisversion 3 KBytes Information speichern, kann um bis zu 48 KBytes erweitert werden und besitzt einen ROM-Programmspeicher von 8 KBytes; der Z80A Mikroprozessor führt eine einfache Operation in ca. 1 Mikrosekunde aus, und er ist außerdem so handlich, daß Sie ihn in jeder Tasche unterbringen können.
Und sein Preis? - Vergleichen Sie mit dem obigen Preis!

Dies ist, neben viel Kreativität der Entwicklungsingenieure des Jupiter ACE, einer dreißigjährigen Entwicklung in der Elektronik zu verdanken, die über gedruckte Schaltungen und Transistoren zum Mikrochip führte, und es möglich macht, heute Computer für Jedermann zu produzieren, wie Ihren Jupiter ACE, zu dessen Erwerb wir Sie beglückwünschen.

KAPITEL 1

AUFSTELLUNG IHRES JUPITER ACE

Zur Anschaffung Ihres neuen Jupiter ACE dürfen wir Sie beglückwünschen und die Hoffnung aussprechen, daß Sie viel Spaß mit Ihrem neuen Computer haben werden.

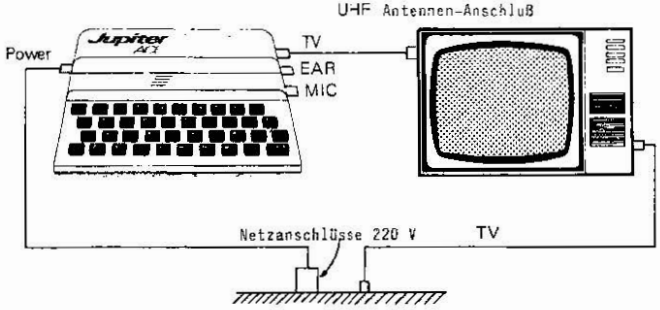
Zusammen mit dem Jupiter ACE erhielten Sie folgendes Zubehör:

1. Dieses Handbuch
2. Ein Netzgerät, das die 220 V Netzspannung auf 9 V transformiert
3. Ein Verbindungskabel zum Fernsehgerät (das längere der beiden Kabel mit je einem Stecker an beiden Enden)
4. Ein Verbindungskabel zum Kassettenrecorder (ein Doppelkabel mit je einem grauen und schwarzen Stecker an beiden Enden)
5. Eine Demonstrationskassette.

Für den Betrieb Ihres ACE benötigen Sie nichts weiter als eine Steckdose (mit 220 V Anschluß), sowie ein Fernsehgerät. Später werden Sie noch einen Kassettenrecorder brauchen, der Anschlüsse für ein Mikrophon und einen Kopfhörer besitzt. Für den Anfang geht es aber auch ohne diesen.

Zuerst schließen Sie bitte das Netzgerät an die Steckdose an. Der kleine Stecker am anderen Ende des Netzgerätes kommt in die Buchse POWER an der linken Seite des Computers (schauen Sie bitte auch auf die Unterseite des Computers, wo alle Anschlüsse markiert sind).
Damit ist der Jupiter ACE angeschlossen und in Betrieb. Nur, ohne Fernsehgerät sehen Sie natürlich nichts davon. An der Rückseite Ihres Fernsehgerätes befindet sich der Anschluß für die Fernsehantenne. Dort hinein kommt das eine Ende des Verbindungskabels (s.Pkt.3). Das andere Ende kommt in den TV-Anschluß des Computers (die größere der drei Buchsen auf der rechten Seite).

Schalten Sie nun Ihr Fernsehgerät ein, drehen Sie den Ton ab und stellen auf UHF Kanal 36 ein. Besitzen Sie ein Gerät mit Stationstasten zur Senderwahl, suchen Sie sich eine aus, schalten auf UHF und drehen solange, bis Sie auf dem Bildschirm ein einheitlich graues Bild haben, mit einem kleinen weißen Viereck unten links - damit ist Ihr ACE fertig angeschlossen, betriebsbereit und mit Ihrem Fernsehgerät in richtigem Kontakt. (Der Anschluß des Kassettenrecorders wird in Kapitel 3 besprochen.)



Ihr Jupiter ACE arbeitet mit der Programmiersprache FORTH. Diese Sprache wurde 1970 von Charles Moore entwickelt und von ihm zunächst zur Steuerung des großen Radioteleskops am Kitt Peak Observatorium verwendet.

Wir haben FORTH für den Jupiter ACE gewählt

- weil es eine sehr schnelle Programmiersprache ist
- weil es sehr wenig Speicherplatz benötigt und Ihnen damit mehr Platz als andere Programmiersprachen für Ihre Programme zur Verfügung stellt
- wegen der besonderen Struktur der Sprache, die es wie kaum eine andere ermöglicht, auch kompliziertere Programme schrittweise aus kleinen, einfachen Programmen aufzubauen.

Falls Sie FORTH bereits beherrschen, werden Sie dieses Handbuch hauptsächlich zum Nachschlagen benutzen. Kapitel 2 sagt Ihnen dann alles über die Eingabe, und Anhang D gibt einen schnellen Überblick über die Besonderheiten von FORTH auf dem Jupiter ACE.

Es ist auch möglich, daß Sie am eigentlichen Programmieren nicht (oder noch nicht?) interessiert sind. Sie wollen also mit bereits fertigen,von uns gelieferten Programmen (der software) arbeiten. Für diesen Fall genügt es, wenn Sie Kapitel 2 und 3 durcharbeiten. Wir hoffen aber, daß Sie im Laufe der Zeit doch auf den Geschmack kommen und später selbst programmieren wollen.

Wenn Sie FORTH noch nicht kennen, vielleicht noch nie mit einem Computer gearbeitet haben, es aber lernen möchten, dann ist dieses Handbuch für Sie geschrieben.

Wir setzen keinerlei einschlägige Kenntnisse voraus und werden Ihnen langsam, Schritt für Schritt, die Elemente der Programmiersprache FORTH vorstellen. Gleichzeitig versuchen wir dabei auch, Ihnen die generelle Arbeitsweise eines Computers zu verdeutlichen.

Besonders ans Herz legen möchten wir Ihnen die Übungen am Ende jedes Kapitels, weil wir überzeugt sind, daß die erlernte Theorie nur in der Praxis erprobt und vertieft werden kann. Es sind dort auch Programme aufgeführt, die Ihnen hoffentlich Spaß machen. Außerdem wird auf interessante Dinge eingegangen, die über das jeweilige Kapitel hinausgehen.

Und nun viel Spaß und guten Erfolg!

KAPITEL 2

EINGABE ÜBER DIE TASTATUR

Ihr Jupiter ACE sollte jetzt betriebsbereit vor Ihnen stehen. Der Fernsehschirm zeigt ein gleichmäßiges graues Bild, unten links befindet sich ein kleines weißes Viereck, die MARKE.

Drücken Sie jetzt wahllos auf einige Tasten. Ihr ACE hat Drucktasten, und mit ein wenig Übung spüren Sie genau, ob Sie die Taste korrekt gedrückt haben oder nicht. Das Ergebnis sehen Sie auf dem Bildschirm.
Vielleicht wird Ihnen das Durcheinander auf dem Bildschirm langsam zu groß? Sie können alles wieder löschen, wenn Sie kurzzeitig den Stecker aus dem Gerät ziehen (in Kürze werden Sie elegantere Methoden kennenlernen).

Wie Sie sehen können, erscheinen die Zeichen, die Sie eintippen, im unteren Teil des Bildschirms. Dieser Teil heißt EINGABEFELD. Aus diesem Eingabefeld holt sich der Computer seine Anweisungen ab. Wenn Sie mehr als eine Zeile vollschreiben (was ganz einfach geht: lassen Sie einmmal eine Taste für ein paar Sekunden gedrückt, dann wiederholt sich das Zeichen fortwährend), rückt die Zeile nach oben und macht Platz für die nächste Eingabe. Das Eingabefeld ist also variabel, es kann sich bei Bedarf nach oben erweitern.
Das Zeilenende brauchen Sie nicht zu beachten, ACE schreibt über die Zeilen hinweg. Das Zeilenende existiert nämlich nur auf Ihrem Bildschirm nicht aber im Innern von ACE.

Bis jetzt erscheinen alle Buchstaben, die Sie eingeben, kleingeschrieben. Wollen Sie einen großen Buchstaben schreiben, drücken Sie (wie bei einer Schreibmaschine) die Taste SHIFT und dazu einen Buchstaben, sagen wir das a, und auf dem Bildschirm erscheint ein A.

Vorne rechts befindet sich (neben der Taste SPACE) die Taste SYMBOL SHIFT. Damit können Sie die Symbole, Rechenzeichen usw. schreiben, die über den Buchstaben und Zahlen auf den Tasten stehen. Drücken Sie zum Beispiel SYMBOL SHIFT und dazu das c, so erhalten Sie ein ?; SYMBOL SHIFT und 1 ergibt das ! usw.

Achtung:

Computer sind völlig fantasielos und äußerst pingelig bei der Verwendung von Zeichen. Von der Schreibmaschine her sind Sie es vielleicht gewöhnt, für die Null einmal 0, einmal den kleinen oder großen Buchstaben o zu schreiben; oder für die Ziffer 1 einmal die 1, einmal das Große I oder das kleine l. Im Umgang mit Computern ist das leider nicht möglich.
Zur besseren Unterscheidung vom Buchstaben o schreiben wir und auch Ihr ACE die Null mit Schrägstrich Ø. Alle zehn Ziffern befinden sich auf der oberen Reihe des Tastatur.

Vielleicht wundern Sie sich, daß der Computer auf das Durcheinander, das Sie eingetippt haben, überhaupt nicht reagiert. Das liegt daran,, daß Sie bisher wirklich nur Zeichen getippt haben, aber noch nichts eingegeben. EINGEBEN geschieht durch die wohl wichtigste aller Tasten, sie heißt ENTER und befindet sich in der zweiten Reihe von unten rechts.
Tippen Sie irgendetwas und drücken dann ENTER. Höchstwahrscheinlich steht jetzt am Anfang der Zeile ein ▯. Damit teilt Ihnen der Computer höflich mit, daß er Ihre Botschaft nicht verstanden hat, gibt Ihnen aber die Möglichkeit, sich zu korrigieren. ▯ heißt also " bitte korrigiern Sie".

Löschen Sie den Bildschirm, indem Sie kurz den Stecker herausziehen und drücken noch einmal ENTER. Die Antwort lautet jetzt OK. Mit OK meldet der Computer " Befehl ausgeführt, bin zu neuen Taten bereit!" In unserem Fall hat der Computer alles brav ausgeführt, was wir ihm aufgetragen haben (nämlich nichts), und meldet sich nach getaner Arbeit bei Ihnen mit OK wieder zurück.

Was versteht ein Computer?

Nun, er versteht wie wir Wörter. Leider keine deutschen Wörter, nicht mal englische. Ihr ACE versteht die Wörter der Programmiersprache FORTH. Um sie vom übrigen Text besser unterscheiden zu können, werden in diesem Handbuch alle FORTH-Wörter im Halbfettdruck ausgedruckt.

Hier das erste FORTH-Wort **VLIST**

Es steht für Vokabelliste. Löschen Sie bitte den Bildschirm und geben Sie **VLIST** ein (d.h. **VLIST** gefolgt von ENTER). Es ist dabei völlig gleichgültig, ob Sie **VLIST** mit großen oder kleinen Buchstaben schreiben. Nur in diesem Handbuch sind FORTH-Wörter immer groß und halbfett gedruckt.

Auf Ihrem Bildschirm steht nun folgendes:

```
VLIST

FORTH  UFLOAT  INT  FNEGATE  F/  F*  F
+  F-  LOAD  BVERIFY  VERIFY  BLOAD  B
SAVE  SAVE  LIST  EDIT  FORGET  REDEF
INE  EXIT  ."  (  [  +LOOP  LOOP  DO  UN
TIL  REPEAT  BEGIN  THEN  ELSE  WHILE
 IF  ]  LEAVE  J  I'  I  DEFINITIONS  V
OCABULARY  IMMEDIATE  RUNS>  DOES>
COMPILER  CALL  DEFINER  ASCII  LITE
RAL  CONSTANT  VARIABLE  ALLOT  C,  ,
 CREATE  :  DECIMAL  MIN  MAX  XOR  AN
D  OR  2-  1-  2+  1+  D+  -  +  DNEGATE
NEGATE  U/MOD  */  *  MOD  /  */MOD  /M
OD  U*  D<  U<  <  >  =  0>  0<  0=  ABS  O
UT  IN  INKEY  BEEP  PLOT  AT  F.  EMIT
 CR  SPACES  SPACE  HOLD  CLS  #  #S  U
.  .  SIGN  #>  <#  TYPE  ROLL  PICK  OV
ER  ROT  ?DUP  R>  >R  !  @  C!  C@  SWAP
 DROP  DUP  SLOW  FAST  INVIS  VIS  CO
NVERT  NUMBER  EXECUTE  FIND  VLIST
WORD  RETYPE  QUERY  LINE  ;  PAD  BAS
E  CURRENT  CONTEXT  HERE  ABORT  QUI
T OK
 ■
```

Das ist die vollständige Liste aller FORTH-Wörter, die ACE von Haus aus versteht, also sein eigenes Wörterbuch. Wie Sie sehen können, handelt es sich dabei im englische Ausdrücke, Abkürzungen, mathematische Zeichen und Kombinationen verschiedener Symbole.

Gegen Ende der Liste finden Sie das Wort **VLIST** wieder (das **VLIST** am Anfang ist etwas anderes: zur Kontrolle schreibt der Computer alles, was Sie ihm eingegeben haben, noch einmal ganz oben auf den Bildschirm, in diesem Falle also **VLIST**).

Das OK am Ende der Liste ist kein FORTH-Wort. Es ist ein interner Ausdruck von ACE und signalisiert " Befehl ausgeführt!"

Sie können auch mehrere Wörter gleichzeitig eingeben, z.B.

VLIST VLIST

Daraufhin schreibt der Computer zunächst eine Kopie Ihres ersten Befehls **VLIST** auf den Bildschirm, führt ihn aus, schreibt also die Liste aller Wörter, und verfährt dann mit dem zweiten **VLIST** genauso; abschließend meldet er sich mit OK zurück.

Unbedingt erforderlich ist der Zwischenraum (Taste SPACE) zwischen den Wörtern. Wir klugen Menschen finden uns noch einigermaßen zurecht, wenn Wörter plötzlichzusammengeschrieben od erf alsch zert eilt werden. Ein Computer ist damit hoffnungslos überfordert. VLISTVLIST ist für ihn ein einziges Wort, das er im Wörterbuch nicht findet, und VLI STVLIST sind zwei Wörter, die er ebenfalls nicht kennt, er antwortet deshalb mit ▮

Dagegen stört das Schrteiben über das Zeilenende hinaus ihn überhaupt nicht, denn wie schon gesagt, das Zeilende existiert nur auf dem Bildschirm, für den Computer ist das gesamte Eingabefeld eine einzige Zeile.

Zusammenfassung

- was immer Sie schreiben, erscheint im Eingabefeld im unteren Teil des Bildschirms
- Buchstaben erscheinen normalerweise als kleine Buchstaben. Großbuchstaben erhalten Sie durch gleichzeitiges Drücken der Taste SHIFT
- ebenso erhalten Sie Satzzeichen und andere Symbole durch Drücken der Taste SYMBOL SHIFT
- der Compuer verfügt über ein eingebautes Wörterbuch mit 142 FORTH-Wörtern, die er sofort verarbeiten kann. Sie können diese Wörter in großen oder kleinen Buchstaben eingeben
- werden mehrere Wörter auf einmal eingegeben, müssen diese durch Zwischenräume (Taste SPACE) getrennt werden
- die Eingabe erfolgt durch die Taste ENTER. Daraufhin nimmt der Computer die Wörter (Befehle) einzeln aus dem Eingabefeld, schreibt sie zu Ihrer Kontrolle oben auf den Bildschirm und führt sie aus
- VLIST ist ein Wort der Programmiersprache FORTH. Auf VLIST schreibt der Computer eine Liste aller FORTH-Wörter aus seinem Wörterbuch
- findet der Computer in seinem Eingabefeld einen Ausdruck, den er nicht versteht, druckt er ▮ an den Anfang der Zeile. ▮ bedeutet:" bitte korrigieren Sie !".

Das Korrigieren von Tippfehlern

Bisher können Sie, wenn Sie sich vertippt haben oder vielleicht vergessen haben, einen Zwischenraum (SPACE) zu lassen, nur alles auf einmal löschen, indem Sie den Netzstecker herausziehen. Das kann natürlich nicht die einzige Möglichkeit sein. In der Tat bietet Ihnen ACE sehr komfortable Korrekturmöglichkeiten.

Wie Sie sicher längst gemerkt haben, zeigt das kleine weiße Viereck, die MARKE, immer genau die Stelle auf dem Bildschirm an, wo das nächste Zeichen erscheint.

Nun gibt es verschiedene Möglichkeiten, diese MARKE zu bewegen. Dazu benützen Sie die Tasten der oberen Reihe, die zusätzlich mit ← ↑ ↓ → gekennzeichnet sind. Drücken Sie, wie bei Großbuchstaben, die Taste SHIFT und dazu die 5 oder die 8. Die Marke bewegt sich in Richtung des Pfeils.

Haben Sie also irrtümlich VIST statt VLIST getippt, so können Sie das leicht korrigieren. Rücken Sie durch SHIFT 5 die Marke um drei nach links, dann haben Sie V IST. Jetzt können Sie das fehlende L eintippen und mit SHIFT 8 die Marke wieder nach rechts rücken, um weiter zu schreiben.

Sehr praktisch ist auch die Taste DELETE (SHIFT Ø). Sie löscht das Zeichen links von der MARKE.

Beispiel:

Sie haben **VLOST** getippt anstatt **VLIST**

rücken Sie die MARKE um zwei nach links **VLO▌ST**

drücken Sie DELETE **VL▌ST**

tippen Sie dass fehlende I **VLI▌ST**

Sie können jetzt ruhig schon ENTER drücken, der Computer kümmert sich nicht darum, wo die Marke steht.

Die Taste SHIFT 6 kann zweierlei bewirken. Angenommen, Sie haben schon mehr als eine Zeile geschrieben, dann springt die Marke durch ↑ senkrecht um eine Zeile nach oben. Befindet sie sich schon in der obersten Zeile, so springt sie an den Anfang der Zeile.
Ebenso funktioniert ↓ (SHIFT 7). Die Marke springt eine Zeile tiefer oder an das Ende der Zeile.

Fast alle Zifferntasten haben in Verbindung mit SHIFT eine zweite Funktion:

DELETE LINE (SHIFT 1) löscht das gesamte Eingabefeld.

CAPS LOCK (SHIFT 2) Feststeller für Großbuchstaben
Die MARKE verändert sich zu C und die Buchstaben werden jetzt groß geschrieben.
Wenn Sie aber ← ↑ ↓ → betätigen wollen, brauchen Sie weiterhin SHIFT.
Nochmaliges Drücken von CAPS LOCK stellt den alten Zustand wieder her.

INVERSE VIDEO (SHIFT 4) alles was Sie schreiben, erscheint jetzt negativ d.h. schwarz auf weißem Grund. Das nochmalige Drücken von INVERSE VIDEO stellt den Normalzustand wieder her.

GRAPHICS (SHIFT 9) verändert die MARKE zu G. Sie können jetzt graphische Zeichen schreiben. Mit den Tasten 1 bis 7 schreiben Sie jetzt die Figuren, die auf den Tasten zu sehen sind. Nochmaliges Drücken ergibt wieder den Normalzustand.

CAPS LOCK, INVERSE VIDEO und GRAPHICS können alle unabhängig voneinander ein-bezw.ausgeschaltet werden.

Drücken Sie zum Beispiel nacheinander:

CAPS LOCKS = Großbuchstaben

INVERSE VIDEO = "negative" Großbuchstaben

CAPS LOCKS = "negative" Kleinbuchstaben

GRAPHICS = graphische Zeichen aber "negativ"

INVERSE VIDEO = die normalen graphischen Muster der Tasten 1-7

GRAPHICS = Normalzustand

Übungen

1. Üben Sie das Korrigieren in allen Variationen.
2. Wir möchten Sie nun endlich davon erlösen, ständig den Netzstecker ziehen zu müssen: was immer Sie getippt haben, DELETE LINE löscht alles auf einmal, DELETE löscht den letzten Buchstaben Ihrer Eingabe. Was der Computer geschrieben hat (z.B.das Wörterbuch) können Sie durch Eingabe des FORTH-Wortes **CLS** löschen!

12

KAPITEL 3

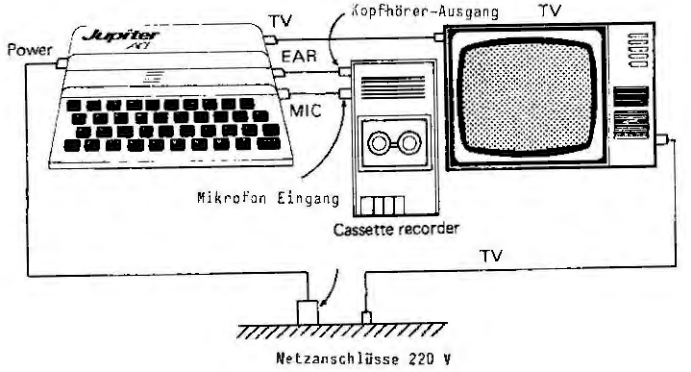
PROGRAMME VON DER KASSETTE LADEN

Falls Sie bereits ACE-Programme auf Kassette besitzen (z.B.Spiele aus unserem Lieferprogramm), erfahren Sie in diesem Kapitel, wie man Programme von Magnetbandcassetten in den ACE lädt.

Sollten Sie noch keine fertigen ACE-Programme haben, können Sie dieses Kapitel vorerst übergehen. Es lassen sich natürlich nur solche Programme laden, die speziell für den ACE geschrieben wurden.

Sie brauchen einen normalen Kassettenrecorder, am besten sogar ein einfaches Gerät, da teure Hifi-Stereoanlagen oftmals die gespeicherten Signale in einer Weise beeinflussen, die der Computer nicht versteht. Der Kassettenrecorder muß eine Mikrofon-Eingangsbuchse und eine Kopfhörer-Ausgangsbuchse besitzen. Die Buchsen müssen zu den Klinkensteckern des mitgelieferten Doppelkabels (schwarz u.grau) passen. Sollte dies nicht der Fall sein, besorgen Sie sich im Radio/TV-Fachhandel einen entsprechenden Adapter.

Verbinden Sie jetzt den Computer und den Recorder mit dem Doppelkabel. Dabei gehört der Kopfhörerausgang zur Buchse EAR des Computers und der Mikrofon-Eingang zur Buchse MIC. Sie können den richtigen Anschluß durch die unterschiedlichen Kabelfarben kontrollieren.



Auf einem Band können mehrere Programme gespeichert sein. Sie sind in einer besonderen Form dort gespeichert, oder wie man in der Fachsprache sagt, codiert, die sich für diese Art der Speicherung eignet.

Jedes Programm hat einen Namen bis zu 1Ø Zeichen Länge, der ebenfalls auf dem Band gespeichert ist.

Wollen Sie jetzt ein Programm laden, gehen Sie wie folgt vor:
Legen Sie die Cassette ein, spulen Sie an den Anfang zurück, oder wenigstens soweit, daß Sie sicher sind, vor dem Programmanfang zu stehen. Hat der Recorder einen Klangregler, so stellen Sie Ihn auf Minimum bezw. Bässe ein, und stellen Sie die Lautstärke auf etwa dreiviertel des Höchstwertes. Geben Sie jetzt in den ACE ein

LOAD Name

(Name des Programms, das Sie laden wollen),
drücken Sie ENTER und schalten Sie den Recorder auf Wiedergabe ein.

Bitte beachten: Der ACE behandelt normalerweise große und kleine Buchstaben gleich, Sie können z.B. ein Wort mit Großbuchstaben eingeben und mit Kleinbuchstaben wieder abrufen. Ein Programm kann vom Band jedoch nur genauso wieder geladen werden, wie es ursprünglich gespeichert war.

Da der Computer verschiedene Programme auf dem Band vorfinden kann, bevor er zu dem gewünschten kommt, schreibt er alle gefundenen Namen von Programmen auf den Bildschirm. Die Anzeige erfolgt in der Form:

Dict.: (Name des Programms)

Wenn Ihr Programm vollständig geladen ist, meldet der Bildschirm OK. Sie können jetzt das Bandgerät abschalten.

Läßt sich ein Programm nicht laden, oder läuft das Band zu Ende, ohne daß das ACE reagiert, sollten Sie

- nachprüfen, ob Computer und Kassettenrecorder richtig und einwandfrei miteinander verbunden sind

- nachprüfen, ob der Programm-Name richtig, auch nach Groß-und Kleinbuchstaben unterschieden, eingegeben ist

- erneut laden mit verschiedenen Einstellungen des Lautstärkereglers

- evtl. den Magnetkopf des Recorders reinigen.

Wenn Sie nicht wissen, welche Programme sich auf einem Band befinden, spulen Sie es auf den Anfang zurück, geben ein

LOAD

drücken ENTER und starten das Band. Der Computer zeigt auf dem Bildschirm die Namen aller gespeicherten Progamme an.

Es können auch mehrere Programme hintereinander in den Speicher des ACE geladen werden, vorausgesetzt, der Speicher ist groß genug.

Manche Programmteile müssen mit BLOAD geladen werden. Es handelt sich dabei um Informationen, die an eine ganz bestimmte Stelle im Speicher geladen werden müssen.
Die übliche Darstellungsform für diesen Fall ist

Ø Ø BLOAD Name

Im Kapitel 14 erfahren Sie darüber mehr.

14

KAPITEL 4

DEFINITION NEUER WÖRTER

Wenn Sie das Wort **VLIST** eingeben, erscheinen auf dem Bildschirm alle Wörter, die der Computer bereits kennt, also sein Wörterbuch, welches er von uns mitbekommen hat.

Beim ersten Mal erscheinen also diejenigen FORTH-Wörter, die fest im ACE gespeichert sind. Damit aber ist der Wortschatz beileibe noch nicht erschöpft, denn Sie können auch eigene Wörter definieren, d.h. neue Begriffe bestimmen. Auf diesem Wege schreibt man ein Programm, d.h. man sagt dem Computer, was und wie er etwas tun soll.

Ein Beispiel:
Sie wollen dem Computer ein neues Wort **HANS** lernen, welches bedeuten soll, ´Führe **VLIST** zweimal aus´. Sie benutzen dabei zwei spezielle FORTH-Wörter, nämlich : (Doppelpunkt) und ; (Semikolon), wie folgt

```
: HANS
  VLIST VLIST
;
```

: ist ein Wort, welches dem Computer mitteilt, daß Sie anschließend ein neues Wort definieren wollen. Danach kommt zuerst sein Name (**HANS**), und anschließend die Definition, die sagt, wie das Wort **HANS** auszuführen ist.

Sie geben also ein : und ENTER.....hoppla! - Entschuldigung, ich vergaß Ihnen zu sagen, daß nach : unmittelbar das neue Wort kommen muß.Sonst erscheint auf dem Bildschirm ERROR 6 (Fehler 6). Schauen Sie doch bei dieser Gelegenheit in die Anlage B am Ende des Handbuches, dort finden Sie die verschiedenen Fehlermöglichkeiten und ihre Bedeutung.

Wann immer Sie ERROR bei der Eingabe bekommen, müssen Sie diese grundsätzlich bei : neu beginnend wiederholen. Also geben Sie erneut ein:

```
: HANS
(an Leertaste denken!)
```

Wenn Sie jetzt ENTER drücken, sagt der Computer noch nicht OK. Damit erinnert er Sie, daß Sie noch mitten in der Definition sind (er sagt aber weinigstens nicht mehr ERROR).

Als nächstes kommt der wichtigste Teil der Definition, er sagt nämlich dem Computer, was er tun soll, wenn Sie das Wort **HANS** später verwenden, nämlich er soll dann zweimal **VLIST** ausführen.

Geben Sie also ein

```
VLIST VLIST
```

und drücken Sie die Taste ENTER..... aber wiederum bekommen Sie noch kein OK vom Computer. Auch schreibt er Gott sei Dank keine lange Liste von Wörtern, denn er weiß, er ist noch mitten in der Definition des Wortes **HANS**.

Schließlich geben Sie ein

```
;
```

was bedeutet, ´die Definition ist zu Ende´. Wenn Sie jetzt nochmals die Taste ENTER drücken, antwortet der Computer (endlich!) mit OK.

Jetzt kennt der Computer das neue Wort **HANS**. Das können Sie auf zwei Wegen überprüfen.

Zuerst geben Sie **VLIST** ein. Sie können sehen, daß das Wort **HANS** am Anfang des Wörterbuches auf dem Bildschirm erscheint.

Zweitens geben Sie das Wort **HANS** ein und können sehen, wie der Computer das gesamte Wörterbuch zweimal hintereinander auf den Bildschirm schreibt, er führt also, getreu Ihrer Definition, **VLIST** zweimal aus.

Haben Sie das Wort **HANS** versehentlich einmal zu oft eingegeben und wollen das Wörterbuch nicht schon wieder auf dem Bildschirm sehen, drücken Sie einfach die Taste BREAK (SPACE mit SHIFT). Sofort stoppt der Computer und zeigt ERROR 3 an (Sie brauchen keinen Fehler in diesem Fall zu suchen).

Wenn Sie also den Computer während einer Verarbeitung stoppen wollen, funktioniert BREAK fast immer. Im Gegensatz zum Ziehen des Netzsteckers löscht BREAK auch nicht die Wörter, die Sie vorher definiert haben.

Unter bestimmten Bedingungen, z.B. wenn der Computer den Kassettenrecorder benutzt, arbeitet auch die Taste SPACE (ohne Umschaltung mit SHIFT) wie BREAK.

Das Wort **HANS**, welches wir in unserem Beispiel gebraucht haben, ist dafür natürlich ein nichtssagendes, ja unsinniges Wort, das niemand in der Praxis für diesen Zweck benutzen würde.

Sie werden aber bald feststellen, daß die beiden FORTH-Wörter : und ; benutzt werden können, um äußerst wirkungsvolle Wörter zu definieren. Deshalb sollten Sie sich diese Wörter gut einprägen.

Wenn Sie also ein neues Wort definieren wollen, benötigen Sie zuerst

:

zweitens, und zwar auf der gleichen Zeile, jedoch mit einem Leerschritt dazwischen, den Namen des neuen Wortes,

drittens die Definition des neuen Wortes (die sagt, wie das neue Wort unter Benutzung bereits vorhandener Wörter gebraucht wird),
und viertens

;

Man nennt dies eine **Doppelpunktdefinition**, weil Sie : benutzt (es gibt auch noch andere Arten von Definitionen, zu denen wir später kommen).

Wir haben Sie bisher das Wort **HANS** auf drei Zeilen definieren lassen, um zu zeigen, wie eine solche Definition arbeitet. In der Praxis können Sie die gesamte Definition in einer einzigen Zeile eingeben:

```
: HANS VLIST VLIST ;
  (auf Leertasten achten!)
```

Natürlich würden Sie in der Praxis auch ein aussagefähigeres Wort anstelle von **HANS** benutzen, zum Beispiel **2VLISTS**.

Nun zu einer Konstruktion, die Definitionen aussagefähiger, interessanter und auch lustiger macht. Sie kann nur innerhalb von Wortdefinitionen benutzt werden und versetzt das definierte Wort in die Lage, eine Nachricht auszudrucken, wenn es aufgerufen wird.

Die Konstruktion besteht aus dem Wort

```
."
```

(Punkt, Anführungsstriche), dann kommt ein SPACE, dann die Nachricht, die geschrieben werden soll, dann " (Ausführungsstriche).

Diese Konstruktion wird oft in Verbindung mit dem Wort

```
CR
```

(carriage return=Wagenrücklauf) gebraucht, welches bewirkt, daß die nächste Nachricht auf einer neuen Zeile erscheint.

Hier ein Beispiel:

```
: BEN
  CR ." ich mag Dich"
;
```

Beachten Sie bitte: wenn Sie vergessen, das" (Ausführungszeichen) einzugeben, erscheint auf dem Bildschirm

?

um Sie zu erinnern, daß Sie etwas vergessen haben. Denken Sie also daran, das ? bedeutet 'möchten Sie etwas ändern oder ergänzen?

Zusammenfassung

FORTH-Wörter : ; CR ."
BREAK
ERROR (Fehleranzeigen)

Übungen

1. Geben Sie bitte ein

```
." Hallo!"
```

Funktioniert nicht? - Wie schon gesagt, können Sie ." nur innerhalb einer Wortdefinition benutzen (bitte sehen Sie unter ERROR 4 im Anhang B des Handbuchs nach).

CR hat diese Einschränkung nicht. Wenn Sie eingeben

```
CR CR CR CR CR
```

werden Sie feststellen, daß jedesmal eine neue Zeile angesteuert wird.

2. Definieren Sie einige Wörter wie

```
: FUSSBALL
  CR ." HSV HAMBURG"
  CR ." wird deutscher Meister"
;
```

und

```
: LANDWIRTSCHAFT
  CR ." großer EG Butterberg"
;
```

Überraschen Sie Ihre Angehörigen und Freunde, indem Sie sie bitten, die Wörter **FUSSBALL** und **LANDWIRTSCHAFT** einzugeben. Und erfinden Sie selbst eigene FORTH-Wörter mit dazu passenden Texten. Vielleicht einen netten Spruch für jeden Ihrer Angehörigen und Freunde, den Sie unter deren Namen definieren.

17

KAPITEL 5

EINFACHE ARITHMETIK

Computer sind dafür bekannt, daß sie sehr schnell rechnen können. Lassen Sie uns dies auf dem Jupiter ACE versuchen.

Geben Sie bitte ein:

```
2 2 + .
 ↑ ↑ ↑
(an Leertasten denken!)
```

Wenn Sie jetzt ENTER drücken, wird der Computer auf dem Bildschirm schreiben ´2 2 + . 4 OK´, was die Kombination Ihrer Eingabe mit der Antwort ´4 OK´ darstellt. Er hat also korrekt 2 und 2 zu 4 addiert.

Beachten Sie bitte, daß Sie ´2 2 +´ anstatt ´2+2´ eingegeben haben, das heißt, Sie sagen ´Nimm 2 und 2 und addiere sie zusammen´ anstatt ´Nimm 2 und addiere 2´. FORTH arbeitet stets auf diese Weise.

Die Grundregel lautet also:

Geben Sie zuerst die Zahlen ein, die Sie wünschen, dann erst geben Sie die gewünschte Rechenoperation an. Dies ist einem Rezept vergleichbar wo zuerst alle Ingredienzien aufgeführt werden, und dann erst die Anweisung folgt, was mit den einzelnen Bestandteilen zu erfolgen hat.

+ und . (Punkt) sind FORTH-Wörter, sie stehen im Wörterbuch, welches Sie durch Eingabe von VLIST auf den Bildschirm holen können.

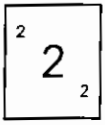
+ addiert zwei Zahlen

. schreibt eine Zahl auf den Bildschirm.

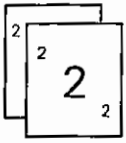
Die Zahl 2 steht nicht im Wörterbuch, aber natürlich wissen Sie und ich und der Computer, daß es eine Zahl ist.

Der Computer speichert die Zahlen, die Sie eingeben, bis Sie ihm sagen was er mit ihnen tun soll. Um Zahlen zu speichern, benutzt Ihr ACE ein kluges Verfahren, welches wir den ´stack´(Stapel) nennen. Der Computer macht dies elektronisch, Sie können sich aber der Einfachheit halber einen Kartenstapel vorstellen, auf dem Zahlen stehen.

Am Anfang ist der Kartenstapel leer: keine Karten, keine Zahlen. Gehen wir zu unserem Beispiel zurück. Wir hatten zuerst die Zahl 2 eingegeben. Der Computer nimmt eine leere Karte, schreibt die Zahl 2 darauf und legt diese Karte oben auf den stack.



Für die zweite Zahl 2 holt er sich ebenfalls eine leere Karte, schreibt die Zahl 2 darauf und legt sie wieder oben auf den stack.



Das FORTH-Wort + besagt ´nimm die beiden obersten Karten vom stack und addiere die darauf stehenden Zahlen. Schreibe die Antwort auf eine neue Karte und lege diese oben auf den stack. Wirf die beiden vorhergehenden Karten weg´.

Das FORTH-Wort . besagt: nimm die oberste Karte des stack, schreibe die daraufstehende Zahl auf den Bildschirm und wirf die Karte anschliessend weg.

+ nimmt immer die zwei obersten Karten des stack, unabhängig wieviele Karten noch darunter liegen.

. benutzt die oberste Zahl des stack, alle anderen bleiben unberührt.

Merke also: Die FORTH-Wörter + und . bearbeiten immer die Zahlen, die oben auf dem stack liegen, also die frischesten Informationen, die auch zuletzt auf den stack gegeben wurden.

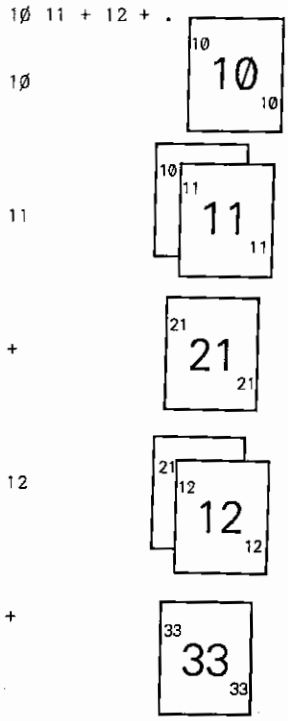
Dies gilt für alle FORTH-Wörter. Es ist einsichtig, daß die zuletzt gekommenen Zahlen, also die neuesten, auch diejenigen sind, welche bearbeitet werden sollen.

Bildlich gesprochen sind die zuletzt gekommenen Zahlen für den Computer immer die frischesten (die älteren sind im stack quasi zugedeckt), und erst wenn die obersten Zahlen verarbeitet sind, erinnert sich der Computer auch wieder der darunter liegenden.

Nun wollen wir einmal drei Zahlen addieren, z.B. 1Ø, 11 und 12. Zuerst müssen Sie 1Ø und 11 addieren. Sie drucken das Ergebnis aber nicht aus, sondern lassen es auf dem stack stehen, um anschließend noch 12 dazu zu addieren.

Geben Sie bitte ein:

1Ø 11 + 12 + .



. zeigt '33' auf dem Bildschirm an

Wenn Sie jetzt vom Addieren genug haben, beschäftigen Sie sich bitte mit den anderen Rechenarten.

- bedeutet Subtraktion. Das Minuszeichen - erzeugen Sie, indem Sie die Taste SYMBOL SHIFT gedrückt halten und dann ´J´ eingeben. Bitte verwechseln Sie das Minuszeichen - nicht mit dem Unterstreichungszeichen _ auf der Taste Ø, das ähnlich aussieht.

* ist das Zeichen für Multiplikation (SYMBOL SHIFT mit B). Bitte verwechseln Sie es nicht mit dem Buchstaben x.

/ ist das Zeichen für Division (SYMBOL SHIFT mit V).

Die Zahlen, die damit verarbeitet werden können, sind ganze Zahlen. Sie können negatives Vorzeichen haben, jedoch können weder Brüche noch Dezimalzahlen verarbeitet werden. (Die Verarbeitung von Dezimalzahlen geschieht unter Verwendung anderer FORTH-Wörter, zum Beispiel **F+** anstelle von + , und **F.** anstelle von . Näheres darüber im Kapitel 15).
Das gleiche gilt auch für das Ergebnis der Division, denn auch hier erscheinen keine Brüche.

Versuchen Sie zum Beispiel 11 durch 4 zu teilen:

11 4 / .

Wenn Sie in Bruchzahlen rechnen würden, wäre das Ergebnis 2 3/4. Da aber nur mit ganzen Zahlen gearbeitet wird, ist die Antwort, die der Computer gibt ´2´.
Ein anderer Weg zur richtigen Antwort wäre ´2 Rest 3´, aber wir haben den Rest nicht abgefragt. Wenn Sie den Rest wissen wollen, dann bewirkt das FORTH-Wort **MOD**, daß der Rest auf dem stack gespeichert wird:

11 4 **MOD** .

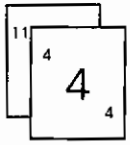
schreibt den Rest ´3´auf den Bildschirm, wenn 11 durch 4 geteilt wird. (**MOD** steht für modulo. 11 modulo 4 ist 3, aber trotz dieses hochtrabenden Begriffs handelt es sich eben nur um den Rest).

Wenn Sie beide, den Quotienten und den Rest sichtbar machen wollen, brauchen Sie das Wort **/MOD**

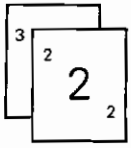
11 4 **/MOD** . .

Die zeigt, wie wirksam es ist, einen stack zu haben, denn **/MOD** speichert zwei Antworten, den Quotienten und den Rest.

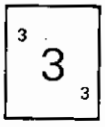
11 4



/MOD



.



schreibt 2
schreibt 3

.

/MOD speichert den Quotienten an oberster Stelle des stack, somit wird dieser auch bei . zuerst ausgedruckt. Als nächstes kommt der Rest.

20

Hier nun eine Liste der meisten FORTH-Wörter, die man für arithmetische Operationen einsetzt. Sie ist nicht vollständig, zeigt aber die Variationsbreite auf, die zur Verfügung steht, bevor man selbst eigene Wörter definieren muß.

Achtung: Diese Wörter arbeiten nur richtig, wen die damit verarbeiteten Zahlen nicht zu groß sind. Siehe Übung 2 dieses Kapitels.

+ - * / MOD und **/MOD** haben Sie bereits kennengelernt.

NEGATE ändert das Vorzeichen der obersten Zahl des stack, d.h. es multipliziert die Zahl mit -1.

1+, 1-, 2+ und **2-** sind speziell definierte FORTH-Wörter, die das gleiche bewirken wie 1 +, 1 - usw. (d.h. mit Leerzeichen), sie verarbeiten aber schneller. Zum Beispiel addiert **1+** eine Eins zu der obersten Zahl des stack.

*/ verarbeitet die drei obersten Zahlen des stack und speichert eine neue dort ab. Es nimmte die drei obersten Zahlen, mulitipliziert die zweite mit der dritten Zahl, und dividiert das Produkt anschließend durch die oberste Zahl des stack.
Zum Beispiel:

6 5 2 */ .

arbeitet wie

6 5 * 2 / .

mit dem Ergebnis (6*5) ÷2= 15.

***/MOD** ist wie ***/**, aber mit einer **/MOD** Operation anstelle von **/**. Es nimmt drei Zahlen vom stack und speichert anschließend zwei dort ab, den Rest und den Quotienten..

MAX und **MIN** nehmen zwei Zahlen vom stack und speichern die größere oder kleinere dort wieder ab.

ABS nimmt eine Zahl vom stack und speichert dort ihren absoluten Wert wieder ab, d.h. die gleiche Zahl, aber ohne Vorzeichen, so daß die dann Null oder positiv ist.

Probieren Sie diese FORTH-Wörter aus, um zu sehen, wie der Computer damit arbeitet.

Zusammenfassung

der stack (Stapel)

FORTH-Wörter +, -, *, /, **MOD, NEGATE, 1+, 1-, 2+, 2-, */, */MOD, MAX, MIN, ABS.**

Übungen

1. Jedes arithmetische Problem kann als Übung auf dem ACE gelöst werden z.B.
   Eine Quarzarmbanduhr kostet 89,5o DM. Wieviel davon ist als Mehrwertsteuer an das Finanzamt abzuführen? (der Preis von 89,50 DM schließt die Mehrwertsteuer von 14 % ein).

   Lösung: wir arbeiten in Pfennigen, um Brüche und Dezimalzahlen zu vermeiden. Der Preis + Mehrwertsteuer ist 114 % des Preises ohne Mehrwertsteuer (der Preis selbst = 100 %, Mehrwertsteuer - 14%, so daß der Gesamtpreis 114 % ausmacht), d.h. um vom Preis ohne Mehrwertsteuer zum Preis mit Mehrwertsteuer zu kommen, multipliziert man mit 114/100. Wir jedoch haben bereits den Preis einschließlich der Mehrwertsteuer, so daß wir den umgekehrten Weg gehen, d.h. wir multiplizieren mit 100/114.

21

Also: Preis ohne Mwst. = 8950 * 100/114. Die Antwort, die wir brauchen, nämlich die Mwst., ist davon 14%, also 8950 * 100/114 * **14/100** = 8950 * 14/114.

Geben Sie also ein:

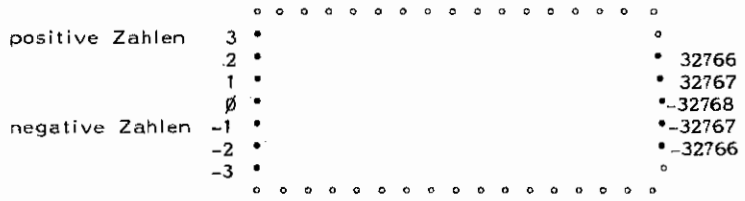
```
8950 14 114 */ .
```

Die Antwort, nämlich wieviel Mehrwertsteuer Sie auf die Armbanduhr bezahlen, ist 1099 Pfennige oder 10,99 DM.

2. Es gibt ein Limit in der Größe der Zahlen, die der Jupiter ACE normalerweise verarbeiten kann. Der Computer sagt Ihnen aber nicht, wenn Sie dieses Limit erreicht haben.
Die größte Zahl ist 32767 und die kleinste -32768.
Tatsächlich aber stellen die verfügbaren Zahlen einen geschlossenen Kreis dar, so daß, wenn Sie

```
32767 1+ .
```

eingeben, Sie -32768 erhalten.



Zahlen außerhalb dieser Reihe werden nicht korrekt eingelesen.
Versuchen Sie

```
32768 .
```

Sie erhalten -32768.

Diese Beschränkung betrifft die meisten Rechenprobleme mi *, weil häufig zwar die beiden miteinander zu multiplizierenden Zahlen selbst klein genug sind, das Resultat jedoch größer als erlaubt. Zum Beispiel:

```
256 256 * .
```

ergibt Ø (die richtige Antwort wäre 65536).

3. Versuchen Sie diese drei Zahlen:

```
256 256 * 256 / .
```

und

```
256 256 256 */ .
```

Die erste Rechnung geht schief, wie wir es in der zweiten Übung erläutert haben. Sie könnten nun erwarten, daß die zweite Rechnung ebenfalls falsch wird: Sie erhalten jedoch die richtige Antwort.
Um zu vermeiden, daß bei Multiplikation mit nachfolgender Division zu große Zahlen, die der Computer nicht mehr verarbeiten kann, entstehen, wurde das FORTH-Wort */ gebildet, welches das verhindern kann.

4. Drücken Sie die . Taste solange, bis nichts mehr auf dem stack ist. Der Computer wird dann eine unsinnige Zahle ausdrucken und anschliessend ERROR 2. Dies bedeutet ´stack underflow´ oder ´es scheint weniger als keine Zahl auf dem stack zu sein´.

´stack underflow´ wird nicht immer sofort entdeckt, weil der Computer dies nur zu bestimmten Zeiten prüft. Zwischen diesen Prüfungen kann es durchaus passieren, daß der Computer in einen Bereich unterhalb des stack gerät, ohne es zu merken.

Das macht aber garnichts (nämlich dem Computer), da er dort einige unsinnige Zahlen vorfindet, mit denen er spielen kann.

Wenn Sie zum Beispiel annehmen, Sie hätten einen leeren stack und geben ein

1 +

wird der Computer 1 zu einer dieser unsinnigen Zahlen addieren.

Da der eigentliche Zweck des + ist, eine Zahl vom stack zu nehmen, bildet sich der Computer ein, daß er die 1 vom stack genommen und somit einen völlig leeren stack hinterlassen hat.

Ein exakt leerer stack ist jedoch nicht ´underflowed´, so daß kein ERROR 2 erscheint. Wenn Sie jedoch erneut + eingeben (nachdem Sie bereits zwei unsinnige Zahlen zusammenaddiert haben), versucht der Computer eine Zahl vom bereits leeren stack zu nehmen, und dies bewirkt dann ´underflow´.

Beachten Sie, daß ERROR stets den stack leert!

5. Versuchen Sie

1 Ø / .

Sie werden überrascht sein, daß die Antwort angeblich -1 sein soll. Dies ist natürlich falsch. Sie dürfen nämlich nicht durch Ø dividieren; tun Sie es trotzdem, erhalten Sie unsinnige Zahlen.

KAPITEL 6

DEFINITION NEUER ARITHMETISCHER WÖRTER

Mit den Wörtern +, -, * usw., die Sie bisher schon kennengelernt haben, verfügen Sie bereits über eine ganze Anzahl geeigneter Bausteine, mit denen Sie neue Wörter bilden können. Hier folgt zum Beispiel ein solches Wort, mit dem Sie Zahlen verdoppeln und anzeigen können:

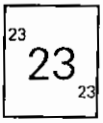
```
: DOPPEL
  2 * .
;
```

Die Zahl, die Sie nun verdoppeln wollen, muß natürlich schon im stack sein, bevor Sie **DOPPEL** verwenden. Geben Sie z.B.folgendes ein:

```
23 DOPPEL
```

dann passiert im stack folgendes:

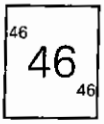
23



DOPPEL führt aus:

2

*



zeigt 46 an

Wie jedes FORTH-Wort greift **DOPPEL** auf den stack zu und holt sich von dort Zahlen. Denken Sie immer daran: ein Wort nimmt Zahlen aus dem stack (das sind die Operanden, mit denen es arbeitet), und legt nach Ende der Bearbeitung andere dort wieder ab (das sind die Ergebnisse).

Es ist aber nicht gesagt, daß die Zahl der Operanden gleich der Zahl der Ergebnisse sein muß.

Zum Beispiel:

+ hat zwei Operanden (die Zahlen die addiert werden sollen) und ein Ergebnis (die Summe).

. hat einen Operanden (die Zahl, die angezeigt wird) und kein Ergebnis (denn die Zahl verschwindet nach der Anzeige aus dem stack).

DOPPEL hat einen Operanden und kein Resultat.

/MOD hat zwei Operanden (Dividend und Divisor) und zwei Ergebnisse (Quotient und Rest).

Die Zahl 2 z.B. könnte man als FORTH-Wort ohne Operanden mit einem Ergebnis (nämlich 2) betrachten.

Mit diesen Beispielen soll unsere Feststellung im Kapitel 5 nochmals untermauert werden:

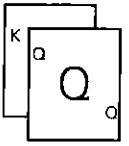
Sammeln Sie zuerst die Zahlen zusammen, mit denen Sie etwas vorhaben, und dann rechnen Sie mit ihnen.

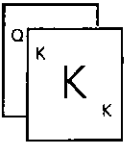
24

Die Zahlen, mit denen man etwas vorhat, sind die Operanden. Zusammengetragen werden sie durch Ablegen im stack.

Einige Wörter haben die Funktion, Zahlen innerhalb des stack zu manipulieren. Die drei einfachsten sind **SWAP**, **DUP** und **DROP**.

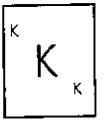
SWAP vertauscht die beiden obersten Zahlen im stack

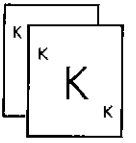
von 

in 

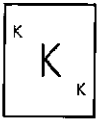
(Richtige Spielkarten haben natürlich aufgedruckte Zahlen und nicht K oder Q, aber da die Zahlen beliebig sein können, stellen wir sie in Form von Buchstaben dar).

DUP dupliziert die oberste Zahl im stack, sie wird also kopiert:

aus 

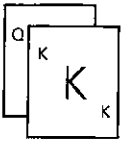
wird 

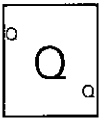
DROP wirft die oberste Zahl des stack weg:

aus 

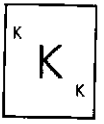
wird NICHTS

Es wird jedoch nur eine Zahl entfernt. Sind zwei oder mehr Zahlen im stack, dann sieht das so aus:

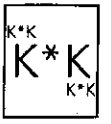
aus 

wird 

Nun folgt ein Wort **SQ**, welches das Quadrat einer Zahl berechnet (d.h. die Zahl wird mit sich selbst multipliziert). Da das Ergebnis nicht angezeigt wird, hat es einen Operanden (die ursprüngliche Zahl) und ein Resultat (ihr Quadrat).

aus 

↑
Operand

entsteht 

↑
Resultat

Nochmals zu Verdeutlichung: der Buchstabe K steht für eine beliebige Zahl, und 'K*K' soll bedeuten, daß K mit sich selbst multipliziert wurde.

Das Wort **SQ** hat folgende Definition:

```
: SQ
  DUP *
;
```

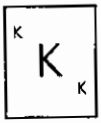
Sie können die Wirkung an verschiedenen Beispielen überprüfen, z.B.

6 **SQ** .

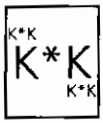
(stellen Sie in der Spielkartenform dar, wie sich der stack im Laufe der Operation **SQ** ändert).

Anstatt weiterhin Bilder von Spielkarten zu rechnen, wollen wir ab jetzt eine vereinfachte symbolische Schreibweise für die Vorgänge im stack benutzen.

Wir ersetzen das Diagramm



wird zu



durch die Zeile

(K - K*K)

Operand ↑ ↑ Resultat

Hat ein Wort mehr als einen Operanden oder mehrere Resultate, werden alle aufgeführt. Für das Wort /MOD z.B. bedeutet das

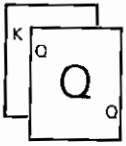
(K, Q - **Rest aus K:Q, Quotient** K:Q)

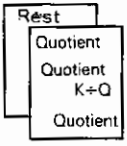
zweiter Op. von oben	oberster Operand	Resultat an der 2.Stelle	Resultat an oberster Stelle

Hier noch eine Erläuterung zur Schreibweise:

Werden Operanden oder Resultate aufgelistet, kommt die Spitze des stack immer zuletzt. In Kartendarstellung sieht das so aus:

Die unterste wird zuerst ausgegeben





Die oberste wird zuletzt ausgegeben

Es ist sehr wichtig, genau zu wissen, welche Operanden jedes Wort im stack erwartet und welche Resultate es schließlich dort hinterläßt.

Es empfiehlt sich deshalb, diese Information als Bestandteil in die Wortdefinition selbst mit einzubauen. Dies geschieht in Form von Kommentaren.

Alles was in runde Klammern ( ) eingeschlossen ist, ist ein Kommentar, dient nur zur Unterstützung des Programmierers und wird vom Computer bei der Ausführung völlig ignoriert.

Hier folgt eine Definition des Wortes **SQ**, die einen solchen Kommentar beinhaltet

```
: SQ
  ( K - K*K )    wird bei Ausführung nicht beachtet
  DUP *
;
```

26

Sie können Kommentare an beliebiger Stelle zwischen dem Namen eines neuen Wortes und dem Semikolon einfügen. Der Inhalt eines Kommentars muß sich nicht auf Erläuterungen zum stack beschränken. Sie können auch Notizen eintragen, die Sie später wieder daran erinnern, was Sie mit dem Wort bezwecken wollten.

Auf die erste runde Klammer (muß ein SPACE folgen, denn das Zeichen (ist ja selbst ein FORTH-Wort mit der Bedeutung 'es folgt ein Kommentar'. Beachten Sie bitte, daß Sie eine schließende Klammer) innerhalb des Kommentars nicht verwenden dürfen, denn sie schließt den Kommentar selbst ab.

So nützlich Kommentare sind, sie verbrauchen viel Speicherplatz:

Sollten Sie also je den Hinweis ERROR 1 erhalten, der besagt, daß der Speicher voll ist, ist es empfehlenswert, als erstes die Kommentare aus Ihren Worten zu entfernen.

Es ist zweckmäßig und üblich, Kommentare solange beizubehalten, bis ein Wort einwandfrei funktioniert. Wenn Sie später Wörter auf Cassette sichern, (Kapitel 14), werden Sie feststellen, daß es praktisch sein kann, zwei Versionen zu speichern.: eine mit Kommentar, als Referenz und Dokumentation, und eine zweite ohne Kommentar für den Gebrauch in anderen Programmen.

Wenn Sie die später folgenden Programmbeispiele aus diesem Buch eintippen, werden Sie wohl kaum die Kommentare mit eingeben, denn die haben Sie ja schwarz auf weiß vor sich liegen.

In Ihren eigenen neuen Programmen, die Sie später machen werden, aber werden Sie den Nutzen von Kommentaren schätzen lernen.

Das Wort (verhält sich genau wie das Wort ." . Beide können Sie nur innerhalb einer übergeordneten Wortdefinition verwenden.

Vergessen Sie die schließende Klammer) , erhalten Sie ?.

Haben Sie beide Versionen von **SQ** eingegeben, dann werden Sie mit VLIST feststellen, daß **SQ** im Wörterbuch zweimal vorkommt. Das soll Sie vorerst noch nicht stören, der Computer verwendet immer die neueste Version, also die, welche Sie zuletzt eingegeben haben. Im nächsten Kapitel lernen Sie, wie Sie die alte Version wieder loswerden können.

Zusammenfassung

FORTH-Wörter nehmen ihre Operanden oben aus dem stack und legen ihre Ergebnisse auch dort wieder ab.

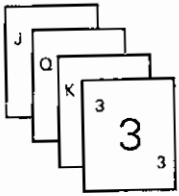
DUP (K - K,K)	(duplicate-kopieren), dupliziert die oberste Zahl im stack
DROP (K -)	(drop-fallen lassen), wirft die oberste Zahl weg, löscht sie.
SWAP (K,Q - Q,K)	(swap-umtauschen), vertauscht die beiden obersten Zahlen
((-)	beginnt einen Kommentar, der mit) abgeschlossen werden muß. Auf den stack hat ein Kommentar keinen Einfluß.

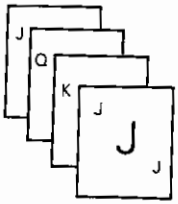
Es gibt noch weitere Wörter, die den stack beeinflussen:

OVER (K,Q - K,Q,K)	kopiert die zweite Zahl auf dem stack nach oben an die Spitze. Der Inhalt des stack wird verschoben.
ROT (K,Q,J - Q,J,K)	vertauscht die oberen drei Zahlen zyklisch und bringt dadurch die dritte Zahl an die Spitze.
PICK (n - K)	nimmt eine Zahl (hier n genannt) von der Spitze des stack, macht eine Kopie der n-ten Zahl von oben und legt diese an der Spitze ab.

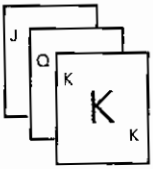
27

PICK ändert zum Beispiel

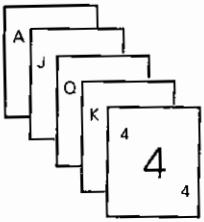


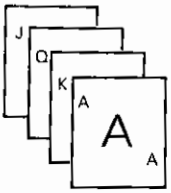


denn J ist die dritte Karte im Stapel

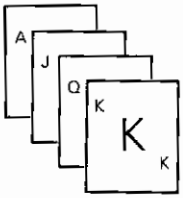


ROLL (n -) nimmt eine Zahl n von oben und vertauscht innerhalb der restlichen Zahlen die ersten n Zahlen zyklisch, wobei die n-te Zahl an die Spitze tritt. Beispiel:





durch Vertauschen der 4 Karten



Übungen

1. Definieren Sie ein Wort, das die Mehrwertsteuer aus einem Verkaufspreis berechnet und als Ergebnis den Preis ohne Mehrwertsteuer hat. (Vergleichen Sie Übung 1 im vorherigen Kapitel. Prüfen Sie anhand der dortigen Zahlen nach, ob Ihr neues Wort das richtige Ergebnis bringt.)

2. Überzeugen Sie sich davon, daß die folgenden Aussagen wahr sind:

1 **PICK** ist dasselbe wie **DUP**
2 **PICK** ist dasselbe wie **OVER**
1 **ROLL** bewirkt nichts
2 **ROLL** ist das gleiche wie **SWAP**
3 **ROLL** ist das gleiche wie **ROT**

Versuchen Sie **PICK** und **ROLL** mit negativen Operanden auszuführen. ....das mag der Jupiter ACE nicht, Sie erhalten ERROR 7.

Andererseits sind **PICK** und **ROLL** relativ unempfindlich, wenn der stack leer ist. Mit **PICK** können Sie niemals ERROR erzeugen, der doch sonst sofort kommt, wenn der stack zu wenig Operanden für ein Wort enthält. **ROLL** spielt mit bis zu 5 unsinnigen Zahlen im stack, bevor es aufgibt und ERROR meldet. Glücklicherweise aber richten die beiden keinen Schaden an. **ROLL** wühlt also nicht etwa aus Mangel an ZAHLEN im Wörterbuch herum.

3. Sie kennen sicher den ´immerwährenden Kalender´, mit dem Sie anhand einiger Tabellen zu einem Datum den jeweiligen Wochentag bestimmen können. Die folgenden Wörter bilden einen solchen Kalender für die Jahre 1901 bis 1999.

DATUM nimmt vom stack den Tag (1 - 31), den Monat (1 Januar, 12 - Dezember), und das Jahr (hier genügen die letzten zwei Stellen) und legt als Ergebnis den richtigen Wochentag in Form einer Zahl ab (1 - Montag, 7 - Sonntag).

FORMAT ist ein Trick, um mit dem Monat Februar fertig zu werden. Das ist notwendig, weil der Februar mit seinen 28 oder 29 Tagen die Berechnung stören würde. Das Wort **FORMAT** tut so, als ob Neujahr am 1. März wäre, und Januar und Februar noch in das alte Jahr gehörten. Die richtige Jahreseinteilung wird gegen diese veränderte Version vertauscht und die Monats-Numerierung entsprechend verändert: Ø - März, 11 - Februar.

JAHR stellt fest, welcher Wochentag der 1. März ist. Es geht von der Tatsache aus, daß der 1. März jeden Jahres um einen, und jedes Schaltjahr um 2 Wochentage verschoben wird. Es zählt die Gesamtverschiebung für das laufende Jahr gegenüber dem 1. März des Jahres 1900 aus (damals war der 1.März ein Donnerstag).

MONAT errechnet zunächst, wieviel Tage zwischen dem 1. März und dem Ersten des eingegebenen Monats liegen: Für März sind das Ø Tage (da ja der März hier der erste Monat des Jahres ist), für April 31 Tage, für Mai 61 Tage usw. Das Ganze geht wieder mit einem Kniff:

Würde man Brüche benutzen (die wir ja noch nicht kennen), könnte man das Ergebnis durch Multiplikation der Monatszahl mit 30,6, Addition von 0,5 und Abrunden auf die nächstkleinere Zahl erhalten; in unserer Ganzzahlarithmetik multiplizieren wir mit 306, addieren 5 und dividieren durch 10.

Im nächsten Schritt wird das Ergebnis von **JAHR** hinzuaddiert, so daß wir die Zahl von Tagen erhalten, um die der Monatserste gegenüber dem 1. März 1900 verschoben ist.

TAG addiert schließlich noch die Tageszahl zum Ergebnis von **MONAT** und wandelt das Ganze dann in den Wochentag um.

Und nun das Programm zum Eingeben:

```
: FORMAT
  ( Monat,Jahr - Monat,Jahr mit Anfangsmonat März)
  SWAP 9 + 12 /MOD
  1-  ROT +
;

: JAHR
  ( Jahr - Zahl der Tage, um die der 1.März gegenüber 1900
           verschoben ist)
DUP 4 / +
;
```

```
: MONAT
  ( Monat,Zahl der Tage - Zahl der Tage, um die der Monats
                          erste gegen den 1.März 1900 ver-
                          schoben ist)
  SWAP 3Ø6 * 5 + 1Ø / +
;

: TAG
  ( Tag,Ergebnis aus MONAT - Wochentag)
  + 2+ 7 MOD 1+
;

: DATUM
  ( Tag,Monat,Jahr - Wochentag)
  FORMAT JAHR MONAT TAG
;
```

Jetzt können Sie Ihren Kalender einsetzen um z.b. festzustellen, an welchem Tag der erste Mensch auf dem Mond landete. Geben Sie ein

```
21 7 69 DATUM .
```

und Sie erhalten ´1´auf dem Bildschirm angezeigt. Neil Armstrong betrat also an einem Montag als erster Mensch den Mond. Wenn Sie es nicht glauben, fragen Sie Armstrong selbst!

Aber natürlich können Sie auch viel näherliegende Daten ermitteln, z.B. wann dieses Jahr Ihre Angehörigen und Freunde Geburtstag haben.

Achtung: Das Programm rechnet für wirklich existierende Tage richtig, mit falschen Eingaben können Sie es aber ´auf's Kreuz legen´. Wenn Sie also fragen, welcher Tag etwa der 29.Februar 1979 war, wird es brav antworten: 4. also Donnerstag, obwohl es diesen Tag überhaupt nicht gegeben hat, denn das Jahr 1979 war ja kein Schaltjahr. Um solche Pannen zu verhindern, müssen Sie wesentlich mehr Aufwand betreiben.

KAPITEL 7

ÄNDERN VON WORT-DEFINITIONEN

In der Programmierung gibt es zwei grundlegende Prinzipien:

1. Programme schreibt man so, daß sie gleich beim ersten Mal richtig arbeiten,

2. Sie funktionieren niemals beim ersten Mal.

Alle neuen Computerprogramme fangen üblicherweise mit dem an, was Programmierer gerne mit dem Wort ´bugs´ umschreiben, mit Fehlern eben.
Einige entstehen aus Unaufmerksamkeit oder aus Tippfehlern, was bewirkt, daß das Programm gar nicht erst zum Laufen kommt. Andere hingegen sind sehr komplizierter Natur, die nur unter bestimmten Umständen zu fehlerhaften Ergebnissen führen können.
Wenn ein Programm Fehler enthält, heißt das noch nicht, daß es schlecht ist und weggeworfen werden muß; wahrscheinlich ist sogar das Meiste in Ordnung, und nur wenige Fehler sind auszubessern.

Nehmen wir ein Beispiel:

```
: EXAKT
  ." 2+2 - "
  2 1 ( ***BUG***) + .
;


: DEMO
  CR ." Hier demonstriert ACE"
  CR ." wie genau er rechnen kann:"
  CR CR CR
  EXAKT
;
```

(Können Sie den Fehler schon erkennen? Rufen Sie bitte nicht gleich bei uns an!)
Wenn Sie jetzt **DEMO** ausführen, werden Sie feststellen, das das Ergebnis falsch ist und Sie müssen den Fehler suchen. Dazu lassen Sie zunächst einmal **EXAKT** laufen: der Fehler steckt in dieser Definition, wie Sie sehen können.
Schauen wir uns also die Definition von **EXAKT** noch einmal an, indem wir eingeben:

```
LIST EXAKT
```

LIST ist ein Wort, welches das folgende Wort nimmt und seine Definition auf dem Bildschirm anzeigt. Programmtechnisch nennt man dies LISTING.
Dabei ist es gleichgültig, in welcher Form Sie Ihre Definition eingegeben haben, **LIST** stellt es immer in einer übersichtlichen Weise dar, die die logische Struktur annähernd erkennen läßt. Es stellt alle Wörter in Großbuchstaben dar, auch wenn Sie sie in Kleinbuchstaben eingegeben haben.
(Erinnern Sie sich an Kapitel 4: ACE kennt keinen Unterschied zwischen Groß-u.Kleinbuchstaben, solange es sich um Computer-Wörter handelt).

Warum, könnten Sie sich fragen, muß man in diesem Fall eingeben **LIST EXAKT** und nicht **EXAKT LIST** ? Schließlich wurde doch in den vorangegangenen Kapiteln immer wieder bis zum Überdruß betont, daß man erst den Operanden (hier: **EXAKT**) eingeben und dann sagen soll, was mit ihm passieren soll (nämlich **LIST**). Nun, **EXAKT LIST** funktioniert einfach nicht:
Der Computer findet zuerst **EXAKT**, führt es aus und zeigt an: " 2+2 = 3". Er findet danach **LIST** und weiß nicht mehr, was er LISTen soll.
Also: erst das Wort **LIST** sagt, daß das <u>nach</u>folgende Wort nicht ausgeführt, sondern gelistet werden soll.

Die allgemeine Regel ist also, daß Zahlen, oder Wörter die Zahlen repräsentieren, vorangestellt werden. Wenn aber ein Wort selbst behandelt werden soll, folgt es nach.

Beachten Sie bitte, daß Sie nur Wörter listen können, die Sie selbst definiert haben.Das hat nichts mit Geheimniskrämerei zu tun, daß Sie mit **LIST** nicht an die von uns bereits fest eingebauten FORTH-Wörter herankommen, sondern diese benutzen Techniken, die **LIST** nicht beherrscht. Wenn Sie es trotzdem versuchen, antwortet der Computer mit ERROR 13.

Kommen wir zurück auf die Fehlersuche in **EXAKT**. Nach intensivem Studium werden Sie feststellen, daß Sie irrtümlich ´1 (***BUG***)´ eingegeben haben anstatt ´2´, und deshalb muß **EXAKT** falsch arbeiten.

Der nächste Schritt ist, **EXAKT** zu korrigieren. Sie brauchen dazu zwei neue Wörter: **EDIT** und **REDEFINE**. Geben Sie bitte ein:

EDIT EXAKT

und Sie erhalten ein LISTING von **EXAKT**, wie vorher bei **LIST**. Diesmal aber im Eingabebereich unten am Bildschirm und mit einem ▯ davor.

Das ▯ ist in Wirklichkeit die MARKE, nur äußerlich etwas verändert, und bedeutet wie üblich ´wollen Sie irgend etwas ändern?´. Weil das Listing im Eingabebereich steht, können Sie es behandeln, als hätten Sie es gerade erst eingegeben. Sie können also mit der MARKE auch jeden Punkt ansteuern und ändern.

Die MARKE läßt sich mit folgenden Tasten bewegen (wobei Sie dabei immer die SHIFT-Tast gedrückt halten müssen, sonst gibt's zusätzlichen Zahlensalat!):

1. ← und → bewegen sie vorwärts und rückwärts innerhalb einer Zeile
2. Mit ↓ und ↑ können Sie die MARKE von Zeile zu Zeile bewegen. Dabei gilt folgendes:
   Steht die MARKE am Anfang der Zeile, wird er mit ↓ an das Ende gebracht. Mit einem weiteren Drücken auf ↓ geht sie an den Anfang der nächsten Zeile. Umgekehrt bewegt ↑ die MARKE vom Zeilenende an den Anfang, bezw. vom Zeilenanfang an das Ende der vorhergehenden Zeile.
3. DELETE LINE (SHIFT und 1) löscht die Zeile, in der die MARKE steht, aber nicht den ganzen Eingabebereich.
4. ENTER speichert wieder den ganzen Eingabebereich, gleichgültig, wo die MARKE gerade steht.

Um zu sehen, wie das geht, wollen wir jetzt **EXAKT** korrigieren. Drücken Sie fünfmal ↓ , dann steht die MARKE am Ende der 3.Zeile, direkt hinter dem Kommentar. Mit DELETE können Sie jetzt Kommentar und 1 löschen. Tippen Sie jetzt die richtige Zahle ´2´ ein. Der Eingabespeicher hat jetzt dieses Bild:

```
: EXAKT
  ." 2+2 - "
  2 2
+ .
;
```

Das genau wollten Sie haben, also drücken Sie ENTER.

Wenn Sie jetzt **EXAKT** laufen lassen, wird es richtig anzeigen: " 2 + 2 = 4 ", aber mit **DEMO** werden Sie eine Überraschung erleben. Es behauptet nämlich nach wie vor: " 2 + 2 = 3 ". **VLIST** zeigt Ihnen, woran das liegt. Es gibt nämlich jetzt zwei Versionen von **EXAKT** im Wörterbuch. Die ältere Version steht ein bißchen weiter rechts, die neue hingegen ganz am Anfang. Tippen Sie nun **EXAKT** ein, sucht der Computer die Definition in seinem Wortschatz, findet die erste und führt sie (richtig) aus.

Wenn Sie jedoch **DEMO** eingeben, sucht der Computer zuerst **DEMO** und danach **EXAKT** und findet prompt die alte (falsche) Fassung. Hier hilft uns nun **REDEFINE**. Geben Sie ein:

REDEFINE EXAKT

Danach können Sie mit **VLIST** feststellen, daß nur noch ein **EXAKT** im Wörterbuch steht. Außerdem arbeiten jetzt beide Wörter richtig.

Vor **REDEFINE** sah unser Wörterbuch so aus:

: **EXAKT** (richtige Definition)
: **DEMO** mit altem **EXAKT**
: **EXAKT** fehlerhafte Definition
weitere Wörter

Mit **REDEFINE** wird zunächst das alte durch das neue **EXAKT** überschrieben

: **EXAKT** richtige Definition
: **DEMO** mit altem **EXAKT**
: **EXAKT** alt, mit Fehlern
weitere Wörter

Dann überarbeitet **REDEFINE** das Wörterbuch und veranlaßt, daß jedes Wort welches das alte **EXAKT** benutzte, jetzt mit dem neuen arbeitet.

: **DEMO** mit neuem **EXAKT**
: **EXAKT** neue, korrigierte Version
weitere Wörter
weitere Wörter

REDEFINE braucht zwei Informationen:

Erstens das alte Wort, welches neu definiert werden soll. Dessen Namen geben Sie direkt nach **REDEFINE** ein.

Zweitens muß **REDEFINE** wissen, wo die neue Version steht. Das ist ganz einfach: Es ist immer das neueste Wort im Wörterbuch. Dabei ist es ganz gleich, welchen Namen dieses Wort hat. In unserem Beispiel ist es zufällig auch **EXAKT**, es könnte aber genau so gut ein Wort mit Namen **BERNHARD** (und noch dazu mit einer ganz anderen Funktion) sein.

Bei der Anwendung von **REDEFINE** ist also Vorsicht geboten: bevor Sie es einsetzen, ist es sehr wichtig nachzuprüfen, ob ein Wort, das Sie am Anfang des Wörterbuches vermuten, auch wirklich dort steht. Sonst steht vielleicht ein völlig anderes Wort am Anfang, und **REDEFINE** macht lauter Unsinn. Die ERROR-Meldung kann man leicht einmal übersehen.

33

Das klingt jetzt furchtbar kompliziert, ist es aber in Wirklichkeit gar nicht. Prägen Sie sich nur die Arbeitsfolge gut ein:

Sie wollen zum Beispiel ein Wort namens **KLEMENS** ändern.

1. Tippen Sie **EDIT KLEMENS** und Taste ENTER
2. Fügen Sie die notwendigen Änderungen ein
3. Drücken Sie ENTER
4. Stellen Sie fest, daß keine ERROR-Meldung vorhanden ist
5. Tippen Sie **REDEFINE KLEMENS** ein

Hier folgen noch einige Tips zur Fehlersuche:

1. Manche Fehler finden Sie einfach durch das Durchsehen des Listings- Ein handschriftliches Listing sollten Sie korrigieren, bevor Sie das Programm eintippen (getreu dem Grundsatz 1 am Anfang dieses Kapitels).
2. Versichern Sie sich, was Sie von jedem Wort erwarten und wie es insbesondere auf den stack wirkt. FORTH ist so konzipiert, daß Sie mit einfachen Wörtern anfangen und daraus weitere, leistungsfähigere Wörter konstruieren können. Haben Sie einmal die einfachen Wörter korrigiert und getestet, können Sie sie unbesehen weiterverwenden.
   Kommentare sind hilfreich, denn ein Kommentar sagt halt " dieses Wort tut dieses und jenes", und Sie testen, ob dies auch wirklich geschieht.
3. Arbeiten Sie wie der Computer und simulieren Sie mit Papier und Bleistift den stack. Notieren Sie Schritt für Schritt, wie der stack sich verändert, während Sie die Definition durcharbeiten. Damit finden Sie meist schon die alltäglichen Fehler.
4. Gerade wenn ein Wort falsche Ergebnisse erzeugt, sollten Sie besonders darauf achten: Sie können wichtige Rückschlüsse auf das tatsächliche Verhalten eines Programms daraus ziehen, auch wenn es nicht das tut, was Sie eigentlich wollten.
   Fehlermeldungen sollten Sie anhand des Anhanges B analysieren und herausfinden, welche Ursache sie haben. So bedeutet z.B. ERROR 2, daß im stack zuwenig Zahlen stehen. Sie müssen also entweder vergessen haben, genügend Zahlen einzustellen, oder zuviele weggenommen haben.

Es gibt noch ein Wort **FORGET**, das Definitionen aus dem Wörterbuch löscht
Aber Vorsicht: Das Wort ist ein wahrer Sprengsatz; es löscht nicht nur das Wort, das Sie dazu eintippen, sondern Alles, was Sie danach noch definiert haben (was also bei **VLIST** noch davor erscheint). In unserem bisherigen Wortschatz würde mit **FORGET EXAKT** auch das Wort **DEMO** verloren gehen. Überlegen Sie also zweimal, bevor Sie **FORGET** einsetzen, es könnte sonst Dinge verschwinden lassen, die Sie noch gut brauchen können.

Wegen der internen Arbeitsweise folgt auf **FORGET** nicht das sonst übliche OK. Machen Sie sich nichts daraus, wenn etwas schiefgeht, meldet sich schon ERROR.

Zusammenfassung

FORTH-Wörter **LIST, EDIT, REDEFINE, FORGET**

Übungen:

1. Verwenden Sie **REDEFINE**, um einen Kommentar in das Wort **SQ** aus dem vorigen Kapitel einzufügen.

2. Angenommen, Sie wollen zwei Versionen von **SQ** gleichzeitig haben. Die eine **SQ** soll die errechnete Quadratzahl auf dem stack ablegen, die andere, nämlich **SQ.** soll es ausdrucken. Überlegen Sie, wie Sie mit **EDIT** aus der bereits gespeicherten Version **SQ** die andere, **SQ.** definieren können. (Sie dürfen dann aber nicht **REDEFINE** verwenden).

3. Ändern Sie unter Verwendung von **EDIT** und **REDEFINE** das Wort **EXAKT** in **TEST** und prüfen Sie mit **LIST** nach, daß in **DEMO** jetzt der Name geändert ist.

4. Geben Sie ein (es erfordert ein wenig Geduld):

```
: EG
  1 2 3 4 . . . .
  ." A"
  ." B"
  ." C"
  ." D"
  ." E"
  ." F"
  ." G"
  ." H"
  ." I"
  ." J"
  ." K"
  ." L"
  ." M"
  ." N"
  ." O"
  ." P"
  ." Q"
  ." R"
  ." S"
;
```

Mit **LIST EG** erhalten Sie ein Listing, das bei 'P' aufhört. **LIST** zeigt nämlich maximal nur 18 Zeilen auf einmal an. Um den Rest auch zu sehen, können Sie eine beliebige Taste betätigen.

Sie werden auch sehen, daß die Zeilen

```
1 2 3 4 . . . .
." A"
```

geändert sind in

```
1 2 3 4 .
. . . ." A"
```

LIST zeigt nicht mehr als fünf Wörter in einer Zeile an und ein ." beendet eine Zeile, ebenso wie ein Kommentar, auch wenn sie weniger als 5 Wörter enthält.

EDIT arbeitet ähnlich. Das Weiterschalten geht allerdings nur mit **ENTER**. Wenn Sie eine Gruppe von Zeilen bearbeitet haben, drücken Sie **ENTER** und der Computer zeigt die nächste Gruppe an.

35

KAPITEL 8

WÖRTER, DIE ZAHLEN BEDEUTEN

Nehmen Sie an, es gäbe eine Zahl, die sich nicht häufig ändert, die Sie sich aber trotzdem nie merken können - beispielsweise Ihr eigenes Alter.

Nehmen wir als Beispiel einen älteren Herrn von 85, da läßt unter Umständen das Gedächtnis schon etwas nach. Er könnte deshalb im ACE sein Alter speichern, indem er ein Wort dafür definiert:

```
: ALTER
  85
;
```

Das funktioniert natürlich. Für solche Definitionen gibt es allerdings eine wesentlich elegantere Methode, und zwar

```
85 CONSTANT ALTER
```

Mit **REDEFINE ALTER** ersetzen Sie nun die bisherige Doppelpunktdefinition. Jetzt können Sie mit **ALTER .** das Alter, nämlich '85' anzeigen.

Der Vorteil dieser Methode besteht darin, daß eine Definition mit **CONSTANT** weniger Speicherplatz braucht als die Definition mit : und die definierte Zahl bei Bedarf auch schneller zum stack bringt.

Allerdings ist es jetzt etwas problematisch, diese Zahl wiederum zu ändern. An seinem Geburtstag müßte der alte Herr eingeben:

```
86 CONSTANT ALTER
REDEFINE ALTER
```

Zum Glück kommt das nur einmal im Jahr vor. Es gibt aber Fälle, in denen solche Zahlen öfter zu ändern sind. Nehmen Sie zum Beispiel den Benzinpreis. Seit dem Ölschock 1973 ist er geklettert und hat sich zeitweise so rasch verändert, daß die Tankwarte Mühe hatten, ihre Tanksäulen immer rechtzeitig umzustellen.

Dann folgte die Energie-Sparwelle, und die Preise veränderten sich - wenn auch nicht ganz so schnell - nach unten.

Für solche Fälle sind Konstante nicht gut geeignet. Deshalb gibt es eine andere Möglichkeit:

```
129 VARIABLE BENZIN
```

Damit wird ein Wort **BENZIN** definiert, das die Zahl 129 (den aktuellen Preis für einen Liter Benzin in Pfennigen) speichert. Es ist zwar etwas umständlicher, diese Zahl wieder herzuholen, dafür können Sie sie aber einfach ändern. **BENZIN** ist eine Variable. Die Zahl 129 ist ihr Wert.

Um diese Zahl auf dem stack abzulegen, verwendet man das Wort **@** (man nennt es "holen"). Mit

```
BENZIN @
```

wird also die Zahl 129 im stack gespeichert. (Kurz zur Erinnerung: Wäre **BENZIN** mit **CONSTANT** definiert worden, hätten Sie das **@** nicht gebraucht). Mit **.** kann man den Wert jetzt natürlich anzeigen.

Seit wir den Benzinpreis vor kurzem definiert haben, ist er schon wieder auf 1,32 DM gestiegen, wir müssen also den Wert von **BENZIN** auf 132 ändern. Das geschieht mit dem Wort **!** (man sagt dazu "speichern")

```
132 BENZIN !
 ↑     ↑
neuer Wert    zu verändernde Variable
```

Jetzt können Sie mit

```
BENZIN @ .
```

den neuen Wert 132 anzeigen.

So bequem können Sie eine gespeicherte Zahl, die mit : oder **CONSTANT** definiert wurde, nicht ändern. Dazu brauchen Sie **REDEFINE**, und eine ganz wichtige Einschränkung für **REDEFINE** ist die Tatsache, daß es nur über die Tastatur angewendet werden kann. Eine Variable wie **BENZIN** dagegen kann auch durch Ereignisse im Programm selbst geändert werden. Das folgende Wort **QBEN** zum Beispiel bildet das Quadrat einer Zahl aus dem stack (indem es das Wort **SQ** aus Kapitel 6 verwendet), und speichert das Ergebnis nach **BENZIN**.

```
: QBEN
  ( K - )
  ( gibt BENZIN den Wert K*K)
  SQ BENZIN !
;
```

Mit **REDEFINE** kriegen Sie das nicht hin!
Versuchen Sie jetzt folgendes:

```
BENZIN
```

Ich weiß nicht, was für eine seltsame Zahl jetzt auf Ihrem Bildschirm erscheint, es ist auf jeden Fall nicht der Benzinpreis (Gott sei Dank!).
Tippen Sie nun diese Zahl, die auf dem Bildschirm gezeigt wird, und nachfolgend @ und . ein, dann erhalten Sie den richtigen Wert von **BENZIN**. Irgendwie muß also diese merkwürdige Zahl, die auf dem Bildschirm angezeigt wird, mit **BENZIN** etwas zu tun haben: Man nennt diese Zahl die Adresse der Variablen **BENZIN**, d.h. **BENZIN** hinterläßt seine Adresse im stack und sagt, "wer meinen Wert braucht, kann ihn hier jederzeit finden". Die Adresse sagt den Wörtern @ und ! wo der Wert zu holen bezw. zu speichern ist.

Dazu müssen wir ein wenig ins Detail gehen. Stellen Sie sich die Elektronik des Jupiter ACE als ein langes Regal mit 65536 Fächern vor. Alle Fächer sind laufend nummeriert mit den Nummern Ø bis 65535, ihren Adressen, ähnlich Hausnummern in einer langen Straße.
In jedes Fach kann nun eine elektronische "Box" gestellt werden, die dann die Adresse des Fachs erhält. Die Box speichert eine Zahl. Es gibt, entsprechend den unterschiedlichen elektronischen Komponenten, verschiedene Arten von Boxen:

|| Eine ROM (=Read Only Memory) - Box ist verriegelt. Sie können ihren Inhalt nicht verändern. Sie ist aber sozusagen durchsichtig, so daß Sie die Zahl feststellen können. Im ACE enthalten die Adressen Ø bis 8191 ROM-Boxen. Sie enthalten die verschlüsselten Befehle, die FORTH zum Laufen bringen, und die fest verdrahteten, immer vorhandenen FORTH-Wörter.

|| Eine RAM (Random Access Memory) - Box ist nicht verriegelt. Sie können ihren Inhalt nicht nur sehen, sondern auch austauschen. Die Adressen 8192 bis 16383 des ACE enthalten RAM-Boxen. Ihre eigenen FORTH-Wörter zum Beispiel sowie das Fernsehbild sind in einer verschlüsselten Form in solchen RAM-Speichern niedergelegt.

|| Andere Fächer wiederum sind leer, wie Bauplätze ohne Häuser, für die schon Hausnummern vergeben sind. Im ACE sind die Adressen von 16384 bis 65535 leer, können aber durch Anschluß geeigneter Zusatzgeräte aufgefüllt werden.

Eine wesentliche Einschränkung all dieser Boxen (wir sollten ab jetzt zutreffender sagen: Speicherplätze) ist, daß die Zahlen, die dort gespeichert werden sollen, zwischen Ø und 255 liegen müssen. Das ist so wichtig, daß man diesen Zahlenwerten sogar einen eigenen Namen gegeben hat: Eine Zahl zwischen Ø und 255 heißt Byte. Jeder Speicherplatz enthält also ein Byte.

Eine normale Zahl kann auf dem ACE zwischen -32768 und 32767 liegen, ist also im allgemeinen durch ein Byte nicht darstellbar. Die Codierung ist aber in Bytes möglich. (Darüber erfahren Sie mehr in Kapitel 17). Damit kann jede Zahl in zwei benachbarten Speicherplätzen abgelegt werden. So hält also z.B. **BENZIN** seinen Wert. Die Adresse von **BENZIN** bezeichnet den ersten der zwei Plätze.

Jetzt können wir also genau sagen, was @ und ! tun:

@ (Adresse - Zahl)
Die Adresse wird aus dem stack geholt. Sie definiert zwei benachbarte Speicherplätze, nämlich den mit der Adresse und den darauffolgenden. Der Inhalt dieser zwei Plätze wird zu einer Zahl entschlüsselt, die im stack abgelegt wird.

! (Zahl - Adresse)
Zahl und Adresse werden vom stack geholt. Die Zahl wird in zwei Bytes verschlüsselt und dann in den Speicherplatz mit der angegebenen Adresse und dem nachfolgenden Platz gespeichert.

Zusammenfassung:

Konstante, Variable

Speicher, ROM, RAM

FORTH-Wörter **CONSTANT VARIABLE @ !**

Übungen:

1. Definieren Sie zwei nützliche Wörter für Ihren eigenen Gebrauch (sie sind nicht fest installiert).

   ? (Adresse -) (genannt "Abfrage")

   zeigt die Zahl an, die in den beiden Speicherplätzen unter der angegebenen Adresse ist. **BENZIN ?** z.B. soll den Wert von **BENZIN** anzeigen.

   +! (Zahl,Adresse -) (genannt "Speicher plus")

   soll ähnlich wirken wie ! , nur anstatt die alte Zahl durch eine neue aus dem stack zu ersetzen, soll ein Wert aus dem stack hinzuaddiert werden. Wenn Sie also z.B. fe ststellen, daß Benzin schon wieder um 6 Pfennige teurer geworden ist, sollen Sie eingeben

   6 BENZIN +!

   Versuchen Sie beide Wörter selbst zu definieren und vergleichen Sie dann Ihre Lösung mit der unseren.

```
: ?
  @
  .
;

: +!
  SWAP OVER @ + SWAP !
;
```

2. Berechnen Sie 2^8 (die "8. Potenz von 2" oder "2 achtmal mit sich selbst multipliziert"). Wieviele verschiedene Bytes gibt es?

   Antwort: 2^8

 38

Es scheint so, als ob die wichtigen Computerbegriffe in Potenzen von 2 ausgedrückt werden, ein Byte ist also nicht so seltsam, wie es zunächst den Anschein hat.
Berechnen Sie auch 2^{15} und 2^{16}. Wo sind Ihnen diese Zahlen schon begegnet? Darüber werden wir im Kapitel 17 mehr sehen.

3. Sind 2 Bytes aus zwei Speicherplätzen gegeben, können Sie diese in folgender Weise zu einer Zahl umrechnen

3.1. Nehmen Sie das Byte aus dem zweiten Platz, multiplizieren Sie es und addieren das Byte aus dem ersten Platz hinzu. |mit 256

3.2. Ist das Ergebnis 32768 oder größer, dann ziehen Sie 65536 davon ab (die Zahl wird negativ).

Versuchen Sie umgekehrt eine Methode zu finden, mit der Sie eine Zahl in 2 Bytes zerlegen können. (3.2. umzukehren ist einfach; um 3.1. umzukehren, dividieren Sie durch 256, so sind Quotient und Rest die beiden Bytes. Denken Sie daran, daß die Zahlen zwischen -32768 und 32767 liegen müssen).

KAPITEL 9

ENTSCHEIDUNGEN

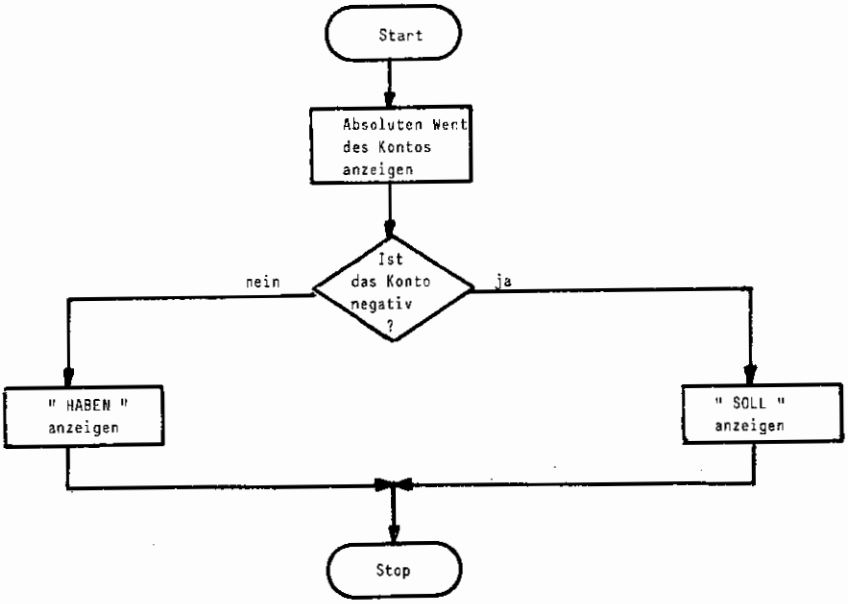
Die Wörter, die wir bisher neu entwickelt haben, sparen dem Anwender zum einen Zeit: um eine Zahl zu quadrieren, braucht er nicht mehr **DUP** * zu tippen, sondern es genügt **SQ**.
Zum anderen - und dies ist noch wichtiger - wird FORTH durch solche Definitionen immer leistungsfähiger.

Mit **SQ** kann es eine Zahl quadrieren, ohne daß sich der Anwender weiter darum kümmern muß, wie dies geschieht.

Diese Wörter sind im Grunde einfache Auflistungen bereits bekannter, anderer Wörter. Das Programm arbeitet sie Schritt für Schritt vom Anfang zum Ende ab und stoppt dann.
Wir brauchen aber in der Praxis Wörter, die unter verschiedenen Bedingungen auch verschieden arbeiten, d.h. Entscheidungen treffen. Damit wollen wir uns jetzt beschäftigen.

Nehmen wir an, Ihr Kontostand bei der Bank sei in der Spitze des stack (oder wie wir in Zukunft fachgerechter sagen wollen: im top of stack, abgekürzt TOS) gespeichert:

Der Bankstand ist positiv oder negativ, je nachdem, ob Sie ein Guthaben besitzen oder Ihr Konto überzogen haben. Mit . wird der Kontostand angezeigt. Hierbei sind Schulden durch ein '-' gekennzeichnet. Allerdings arbeitet die Bank nicht so, sondern sie gibt den absoluten Betrag des Kontos an (also den Betrag ohne Vorzeichen) und ergänzt je nach Situation SOLL oder HABEN. Einen solchen Vorgang kann man mit einem Flußdiagramm verdeutlichen:



40

Wie liest man ein solches Flußdiagramm ?

Sie beginnen bei " Start" und folgen den Pfeilen, bis Sie bei " Stop" angelangt sind. Treffen Sie dabei auf ein rautenförmiges Kästchen, verzweigt sich der Weg. Sie haben nun die Wahl, welchen Weg Sie folgen wollen, müssen also eine Entscheidung treffen.

Teile dieses Flußdiagramms können Sie bereits in FORTH übersetzen:

Start	wird zu	: KONTO (Bankstand -)
Absoluten Wert des Kontos anzeigen	wird zu	ABS . (s.Kapitel 4)
" HABEN" anzeigen	wird zu	." HABEN "
" SOLL" anzeigen	wird zu	." SOLL "
Stop	wird zu	;

Bleibt jetzt noch das Problem, das Entscheidungskästchen zu übersetzen. Hierfür brauchen Sie einige neue FORTH-Wörter: IF, ELSE und THEN.

(Dies sind übrigens drei original englische Wörter: IF heißt ´wenn´, THEN heißt ´dann´, ELSE heißt ´sonst´).

Doch zunächst einmal die Definition für unser Wort KONTO:

```
: KONTO
  ( Bankstand - )
  DUP ABS . Ø<
  IF
    ( Wenn Kontostand negativ)
    ." SOLL "
  ELSE
    ( Wenn Konto Ø oder positiv)
    ." Haben "
  THEN
;
```

IF trifft eine Entscheidung zwischen zwei Wegen: der eine führt von IF nach ELSE, der andere von ELSE nach THEN. Bei THEN treffen sich die Wege wieder.

Entscheidungsgrundlage für IF ist die Zahl am TOS (top of stack = Spitze des stack), welche anschließend verloren geht. Diese Zahl am TOS heißt deshalb auch Bedingung. Ist die Bedingung nicht erfüllt, folgt das Programm dem Weg ELSE - THEN; ist sie erfüllt, dem Weg IF - ELSE.

Machen wir uns die Funktionen der drei Wörter nochmals klar:

IF (=wenn) die Bedingung am TOS erfüllt ist, folge diesem Weg (der mit IF beginnt.
ELSE (=sonst) d.h. wenn sie nämlich nicht erfüllt ist, folge diesem Weg (beginnend mit **ELSE**).

THEN (=dann) fahre hier fort.

Anmerkung: Wenn Sie mit der Computersprache BASIC vertraut sind, dann müssen Sie hier ein wenig umdenken.

Im Allgemeinen wird ein eigenes Wort definiert, das die Bedingung prüft und danach auf dem stack eine ´1´ ablegt, wenn sie erfüllt ist, und eine ´Ø´, wenn sie nicht erfüllt ist. Das Ergebnis einer solchen Prüfung, nämlich ´1´ oder ´Ø´, nennt man einen MERKER (flag). IF benutzt dann diesen MERKER für die weitere Entscheidung.

Wir werden im folgenden mit einigen, Ihnen vielleicht nicht so vertrauten Begriffen arbeiten, die hier kurz erläutert werden sollen:

- Ist eine Bedingung erfüllt, so sagt man von ihr: sie ist "wahr". Der entsprechende MERKER ist die ´1´.

- Ist die Bedingung nicht erfüllt, so heißt sie "falsch". Der zugehörige MERKER ist die ´Ø´.

Es ist vielleicht etwas verständlicher, wenn Sie gedanklich das Wort "Bedingung" durch den Begriff "Behauptung" oder "Annahme" ersetzen. "Wahre" oder "falsche" Behauptungen sind in unserem Sprachgebrauch ja durchaus geläufig.

Das Zeichen < bedeutet "kleiner als". Stellt sich einer vor Sie hin und behauptet "2 < 3", dann nicken Sie weise mit Ihrem Haupt und sagen: "Das ist wahr"; käme er aber an mit der Behauptung "3 < 2", würden Sie voll Entrüstung antworten: "Das ist falsch".

Zurück zu unserem Beispiel:Wir haben bereits eine solche Prüfung verwendet, nämlich Ø< . Hier soll folgende Anweisung ausgeführt werden: "Nimm die oberste Zahl aus dem stack und stelle fest, ob sie negativ ist".

Konto negativ bedeutet Annahme Ø< trifft zu,
danach: 1 auf den stack (= Aussage wahr!)
und: IF geht den Weg IF - **ELSE**.

Konto positiv bedeutet: Annahme Ø< trifft nicht zu,
danach: Ø auf den stack (= Aussage falsch!)
und: IF geht den Weg **ELSE** - **THEN**.

Wir haben mit **DUP** angefangen, sonst wäre nach der Anzeige des absoluten Wertes der Kontostand verloren gegangen und der Vergleich Ø< nicht mehr möglich gewesen.
Gehen Sie **KONTO** nochmals durch und zeichnen Sie die Entwicklung der Daten im stack auf. Tun Sie das ruhig zweimal, und zwar mit einem positiven und einem negativen Anfangswert.

FORTH kennt noch weitere Prüfwörter:

= (K,Q, - Merker) nimmt die obersten beiden Zahlen vom stack und prüft, ob sie gleich sind.

< (K,Q - Merker) nimmt die obersten beiden Zahlen vom stack und prüft, ob die zweite (K) kleiner ist als die oberste (Q). Die Bedingung ist nur erfüllt ("wahr"), wenn sie kleiner ist, nicht wenn beide Zahlen gleich sind.

> (K,Q, - Merker) nimmt die obersten beiden Zahlen vom stack und prüft, ob die zweite (K) größer ist als die erste (Q).

Ø= (K - Merker) nimmt die oberste Zahl und prüft, ob sie Ø ist.

Ø< (K - Merker) nimmt die oberste Zahl und prüft, ob sie negativ ist (kleiner als Ø, nicht Ø selbst.

Ø> (K - Merker) nimmt die oberste Zahl und prüft, ob sie positiv ist (größer als Ø, nicht gleich Ø.

Immer gilt: Ein Prüfwort hinterläßt eine 1 im stack, wenn die Bedingung erfüllt (=wahr), und eine Ø, wenn die Bedingung nicht erfüllt (=falsch) ist.
Die Wörter Ø=, Ø>, und Ø< können natürlich auch durch Ø =, Ø > und Ø < ersetzt werden.

In einem Entscheidungsvorgang kann der ELSE-Zweig entfallen, wenn bei negativem (falschem) Prüfergebnis (im stack ist Ø) nichts besonderes geschehen soll.
Diesen Vorgang könnte man dann so interpretieren:

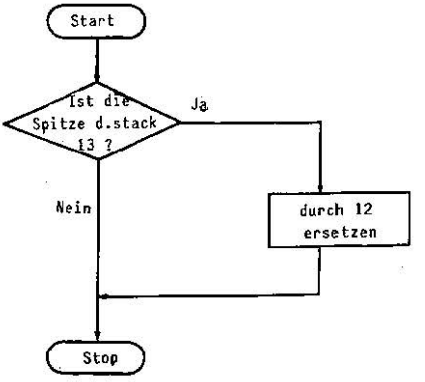
Wenn (IF) die Zahl auf dem stack wahr ist, folge dem IF-Weg.

Dann (THEN) fahre in jedem Fall hier fort.

Im folgenden Beispiel soll eine eingebene Zahl überprüft werden. Hat Sie den Wert 13, wird sie durch 12 ersetzt. Bei anderen Werten passiert nichts

```
: DREIZEHN
  (K - Q)
  DUP 13 =
  IF
     DROP 12
  THEN
;
```

Da für andere Zahlen als 13 keine Aktivität notwendig ist, brauchen Sie auch ELSE nicht. Dazu das Flußdiagramm:



Ein Wort, dessen Nutzen Sie möglicherweise erst später entdecken, ist **?DUP**.

?DUP dupliziert den TOS, aber nur dann, wenn er ungleich Ø ist
(K - K,K) wenn K ungleich Ø
(Ø - Ø)

Nehmen Sie an, Sie brauchen ein Wort, das den TOS nur ausdruckt, wenn er nicht Ø ist. Ohne **?DUP** würde das so aussehen

```
: ?
  DUP
  IF
    .
  ELSE
        DROP
  THEN
;
```

Mit **?DUP** geht es viel einfacher

```
: ?
  ?DUP
  IF
    .
  THEN
;
```

Arbeiten Sie beide Wörter durch und prüfen Sie, was im stack geschieht

Zusammenfassung:

Flußdiagramme

Prüfwörter, MERKER "wahr" und "falsch"

FORTH-Wörter: **IF, ELSE, THEN,** =, <, >, Ø=, Ø<, Ø>, **?DUP**

Übungen:

1. Eine Ø im stack wird durch Ø= in 1 umgewandelt. Ebenso wird 1 zu Ø. Mit Ø= kann man also ein Prüfergebnis umkehren, aus "wahr" wird "falsch". Wollen Sie mit **IF** und **THEN** nur dann etwas tun, wenn das Ergebnis einer vorherigen Prüfung negativ ("falsch") war,
   dann drehen Sie es mit Ø= um und verwenden **IF, THEN** ohne **ELSE**.

2. Der Vergleich zweier Zahlen auf "größer als" oder "kleiner als" wird "falsch", wenn die beiden Zahlen gleich sind. Definieren Sie ein Prüfwort Ø>=, das gleiche und größere Zahlen als "wahr" akzeptiert.

3. Was geschieht, wenn Sie IF, THEN und ELSE außerhalb einer Wortdefinition verwenden? Der Computer meldet ERROR 4. Er will IF verarbeiten, weiß zu diesem Zeitpunkt aber nicht, wo ELSE und THEN zu finden sind. Anders innerhalb einer Wortdefinition: Bei der Übernahme ins Wörterbuch werden die verschiedenen Wörter (als Adressen) erkannt und gespeichert.

4. Versuchen Sie, ein Wort mit IF, ELSE und THEN in der verkehrten Reihenfolge zu definieren. Machen Sie die Definition nicht zu groß, es wäre schade um die Mühe. Der Computer mag das nämlich nicht und läßt mit ERROR 5 das ganze Wort verschwinden.

5. Unter der Fakultät einer Zahl versteht man das Produkt 1*2*3*..... bis zu dieser Zahl. Die übliche Schreibweise dafür ist die Zahl, gefolgt von einem Ausrufungszeichen

```
1! = 1
2! = 1*2 = 2
3! = 1*2*3 = 6
4! = 1*2*3*4 = 24
usw.
```

Auch Ø! wird als 1 definiert. Für diese Festlegung gibt es durchaus mathematische Begründungen.

Geben Sie das Wort FAK ein:

```
: FAK
  (n - n!)
  ?DUP
  IF
      DUP 1- FAK *
  ELSE
      1
  THEN
;
```

Wir verwenden hier eine ganz raffinierte Technik, die sogenannte Rekursion, d.h. FAK benutzt sich selbst! Wir können so verfahren, weil

```
2! = 2 * 1!
3! = 3 * 2!
4! = 4 * 3!
usw.
```

ist.

Um also die Fakultät einer Zahl zu berechnen, ermitteln wir zunächst die Fakultät der vorhergehenden Zahl und multiplizieren diese dann mit der Gegebenen. Das gleiche passiert mit den Fakultäten der kleineren Zahlen, bis Ø erreicht ist. Da Ø! = 1 ist, brauchen wir

```
IF....ELSE 1 THEN
```

6. Eine allgemeine Form der Rekursion liegt vor, wenn mehrere Wörter sich gegenseitig verwenden. FORTH ist zwar für eine solche Anwendung nicht entwickelt, sie kann aber trotzdem durchgeführt werden. Das Problem liegt darin, daß das zuerst definierte Wort die nachfolgenden nicht benutzen kann. Sie sind im Wörterbuch noch nicht definiert.Der Computer reagiert mit einem ?

45

Um dies zu lösen, müssen Sie mit **REDEFINE** arbeiten. Wollen Sie z.B. A und B definieren, die sich gegenseitig verwenden, müssen Sie

1. ein Scheinwort (Dummy) A für die Verwendung in B definieren

```
: A
;
```

2. B endgültig definieren. Diese Definition würde zwar nicht richtig funktionieren, weil A ja ein Dummy ist, Sie können aber B richtig eintasten.

3. A endgültig eintasten

4. Dann

```
REDEFINE A
```

eingeben, so daß das Dummy durch die richtige Definition ersetzt wird.

7. Das Vorzeichen einer Zahl soll "1" sein, wenn die Zahl positiv ist; "Ø" wenn sie Null, und "-1" wenn sie negativ ist.

Schreiben Sie ein FORTH-Wort **SGN**, um eine Zahl im stack durch diese Vorzeichenbedeutung zu ersetzen.

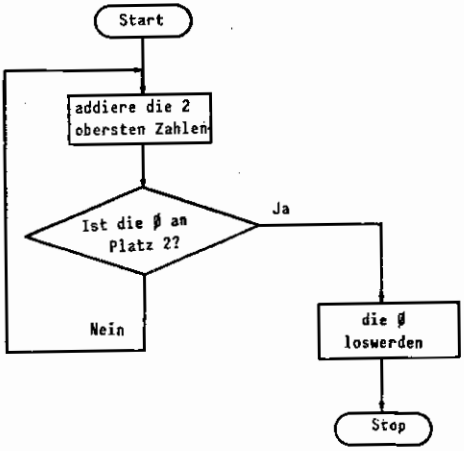
KAPITEL 10

WIEDERHOLUNG

Bisher haben wir noch keine Möglichkeit kennengelernt, Teile eines definierten Wortes mehr als einmal zu benutzen. Der Computer arbeitet die Definition Schritt für Schritt ab, überspringt unter dem Einfluß von IF, ELSE und THEN bestimmte Abschnitte und kommt zum Ende.

Was ist zu tun, wenn z.B. eine unbekannte Anzahl von Zahlen zusammenaddiert werden soll? Eine Grenze muß eingebaut werden, damit der Computer weiß, wann er aufhören soll. Sagen wir ihm also, er soll Zahlen solange addieren, bis er eine Ø erkennt. Wir legen zunächst eine Ø im stack ab und packen alle zu addierenden Zahlen obendrauf.

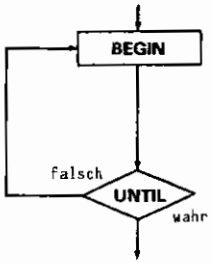
Es sollen wenigstens zwei Zahlen addiert werden: Wir addieren die ersten beiden Zahlen zusammen und prüfen noch, ob die Ø inzwischen schon auf den zweiten Platz nachgerückt ist. Ist dies der Fall, müssen wir sie noch loswerden und sind fertig; andernfalls addieren wir weiter. Im Flußdiagramm sieht das so aus:



Dazu das FORTH-Wort:

```
: ++
  ( addiert Zahlen im stack zusammen bis Ø auftritt)
  BEGIN
    + ( Ø oder nächste Zahl,aufgelaufene Summe)
    OVER Ø=
  UNTIL
  SWAP DROP
;
```

Hier ist **UNTIL** das entscheidende Wort. Sie können zwischen **BEGIN** und **UNTIL** einfügen, was Sie wollen, es muß aber zum Schluß immer eine Bedingung (wahr oder falsch) auf dem stack abgelegt sein, auf Grund der **UNTIL** entscheiden kann. Ist die Bedingung wahr, dann wird die Wiederholung beendet und der Computer bearbeitet den nachfolgenden Abschnitt. Ist sie falsch, springt das Programm zurück zu **BEGIN**



Wie sieht das bei unserem Wort ++ aus? Bei **BEGIN** wird unterstellt, daß der stack die bisher aufgelaufene Summe und die nächste zu addierende Zahl enthält. Erst danach folgt die Ø, die das Programm beendet, oder eine weitere Zahl. (Das geht schief, wenn Sie nicht mindestens zwei Zahlen ungleich Ø vorrätig haben. Wir werden diesen Fehler später beheben).

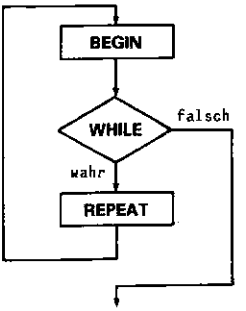
+ addiert die beiden oberen Zahlen und legt die Summe am TOS ab. **OVER** kopiert die nächste Zahl an die Spitze und wir können mit Ø= prüfen, ob es die Ø ist. Wir verwenden hier Ø=, weil eine Zahl ungleich Ø das Prüfergebnis "falsch" bringt und den MERKER "Ø" auf dem stack ablegt. Dadurch wird **UNTIL** veranlaßt, zu **BEGIN** zu verzweigen, und die nächste Addition kann folgen. Wird die Zahl Ø erkannt, geht das Programm weiter, wirft die Ø weg und läßt nur noch die Summe im stack stehen.

Ein Nachteil von **UNTIL** ist sicher, daß der Computer den Abschnitt zwischen **BEGIN** und **UNTIL** wenigstens einmal durchlaufen muß. Das ist auch einleuchtend, denn bei **BEGIN** ist ja noch keine Prüfung und Entscheidung möglich.

Es gibt aber eine Möglichkeit, dieses Problem zu umgehen, indem man die Wörter **WHILE** und **REPEAT** statt **UNTIL** verwendet. Die Schleife hat dann die Form

```
BEGIN
  |
  |
WHILE
  |
  |
REPEAT
```

Hier entscheidet **WHILE**, ob der Abschnitt bis **REPEAT** durchlaufen wird (das ist der Fall, wenn es auf dem stack die Aussage "wahr" findet), oder ob die Schleife verlassen wird.



Es gibt zwei wesentliche Unterschiede zwischen **UNTIL** und **WHILE**..... REPEAT:

1. **UNTIL** beendet die Wiederholung, wenn es die Aussage "wahr" findet, **WHILE** beendet, wenn es "falsch" erkennt.
2. Bei Verwendung von **WHILE** gibt es einen Programmabschnitt (**WHILE-REPEAT**), der nie durchlaufen werden muß, wenn **WHILE** ganz zu Anfang die Aussage "falsch" erkennt.

Damit kommen wir zu einer besseren Version von ++ :

```
: +++
  ( addiert Zahlen im stack bis Ø erkannt wird)
  Ø ( als aufgelaufene Summe bei Anfang)
  BEGIN
    ( aktuelle aufgelaufene Summe, nächste zu add.Zahl)
    SWAP ?DUP
  WHILE
    +
  REPEAT
;
```

Sofort nach **BEGIN** steht die aufgelaufene Summe im TOS, die nächste zu addierende Zahl folgt direkt danach. Wir bringen sie in den TOS, prüfen, ob sie Ø ist und addieren sie, wenn sie nicht Ø ist (d.h. wenn die Aussage "wahr" ist).
Ist sie Ø, gehen wir zum Ende. **?DUP** ist dafür ein sehr nützliches Wort.

+++ verhält sich auch richtig, wenn nur eine oder gar keine zu addierende Zahl im stack ist. Prüfen Sie es nach!

Die beiden Arten, die mit **BEGIN** anfangen, wiederholen eine Schleife solange, bis eine einprogrammierte Bedingung eintritt, die das Ende veranlaßt. Es gibt noch einige andere Arten von Schleifen. Sie beginnen mit dem Wort **DO**, durchlaufen die Schleife mit einer vorgegebenen Häufigkeit und verwenden dazu einen Zähler.

Die einfachste Form ist

```
...DO....LOOP....
```

als Bestandteil einer Wortdefinition.

49

DO nimmt zwei Zahlen vom stack, die bestimmen, wie oft die Schleife durchlaufen wird. Die oberste Zahl bestimmt, wo der Zähler beginnt, die zweite Zahl setzt die Grenze: Die Schleife ist beendet, wenn der Zähler diese Zahl erreicht hat.

In der Folge

```
6 3 DO....LOOP
```

wird der Bereich zwischen **DO** und **LOOP** 3 mal durchlaufen. Der Zähler beginnt bei 3, hat in der zweiten Runde den Wert 4 und in der dritten Runde 5. Beachten Sie, daß der Zähler die Grenze nicht erreicht, sie muß immer um 1 höher sein als der Endwert, den der Zähler erreichen soll. **LOOP** zählt 1 zum Zähler hinzu, prüft, ob der Endwert erreicht ist und springt zurück zu **DO**, wenn dies nicht der Fall ist.

Den "Zählerstand" können Sie erfahren und auf dem stack ablegen, wenn Sie das Wort **I** verwenden.

Wir definieren jetzt das Wort **NR**, das uns alle Zahlenwerte, die der Zähler während der Ausführung hat, anzeigt:

```
: NR
  ( Grenze - )
  Ø
  DO
   I . ( druckt den Zähler aus)
  LOOP
;
```

```
3 NR
```

zeigt an: Ø, 1, 2

```
1 NR
```

zeigt nur Ø an.

Versuchen Sie Ø **NR** und -1 **NR**. Sie erhalten in beiden Fällen die Anzeige Ø. Daraus lassen sich zwei Regeln ableiten:

1. Wie bei **BEGIN...UNTIL** wird auch bei **DO...LOOP** die Schleife mindestens einmal durchlaufen, gleichgültig wie Anfangs- und Endwert heißen (deshalb zeigt **NR** auch die Ø an).
2. **LOOP** beendet den Rücksprung, wenn der Zähler (nach der Addition von 1) gleich oder größer als der Endwert geworden ist.
   Für -1 **NR** heißt das: Der Endwert ist -1, der Zähler geht nach der Anzeige 'Ø' auf 1, ist also größer als -1, das Programm wird beendet.

LOOP erhöht den Zähler immer um 1. Es gibt aber eine Variante **+LOOP**, die die oberste Zahl aus dem stack nimmt und zum Zähler addiert (diese Zahl heißt step = Schritt). Die Regeln 1 und 2 gelten auch hier, es sei denn, der step ist negativ. Dann nämlich hört **+LOOP** auf, wenn der Zähler gleich oder kleiner als der Endwert ist. Geben Sie ein:

```
: COUNTDOWN
  -1 1Ø
  DO
   I . -1
  +LOOP
```

Ohne die Sonderregelung für einen negativen step würde **COUNTDOWN** nur 1Ø anzeigen und dann aufhören.

Es können mehrere dieser Schleifen ineinander verschachtelt werden. Sie laufen dann gleichzeitig ab, wie im folgenden Wort **STERNE**. Dieses Wort gibt ein Dreieck aus Sternen aus. Mit Rücksicht auf die Arbeitsweise von **LOOP** legen wir noch eine 1 zu der eingegebenen Zahl zu.

```
: STERNE
  ( Anzahl der Reihen
  CR 1+ 1
  DO
   I Ø
   DO
    ." *"
   LOOP
   CR
  LOOP
;
```

Jetzt laufen zwei Zähler zugleich, ein langsamer für die Reihen und ein schneller für die Sterne in jeder Reihe. Es stellt sich die Frage, welchen Wert I in diesem Fall darstellt. Generell gilt: Laufen gleichzeitig zwei oder mehr Schleifen ab, entspricht I dem Zähler der innersten oder "kleinsten" Schleife. Das ist die Schleife, die in einem Listing am weitesten rechts steht. Der Zähler der nächstgrößeren Schleife kann mit einem Wort J dargestellt werden:

```
: STERNE
  ( Zahl der Reihen)
  CR 1+ 1
  DO                          |                  Nur Reihenschleife
    I Ø                       |
    DO        | Stern-        | Reihen-          Beide Schleifen
     ." *"    | zähl-         | zähl-
    LOOP      | Schleife      | Schleife
    CR                        |
  LOOP                        |                  Nur Reihenschleife
;
```

Bei Ausführung der Stern-Zählschleife (z.B. in der Zeile ." *") sind beide Schleifen aktiv. Hier würde I den Stern-Zähler und J den Reihenzähler ausgeben. In der übrigen Reihen-Zählschleife (also in den Reihen I Ø und CR) ist nur der Reihenzähler aktiv. I repräsentiert den Reihenzähler, J enthält Unsinn. I kann also in verschiedenen Programmteilen durchaus verschiedene Zähler wiedergeben.

Ein anderes Wort I' legt den Grenzwert der innersten Schleife auf dem stack ab.

Beachten Sie bitte, daß I, I' und J nur in der Wortdefinition verwendet werden können, die auch das ihnen entsprechende **DO...LOOP** enthält.

Alle die Strukturen, die Sie im Kapitel 10 kennengelernt haben, können Sie verwenden, so oft Sie wollen. Es gibt nur eine Einschränkung:
Wenn sie sich überschneiden, müssen sie ineinander verschachtelt werden.

Lassen Sie uns an den folgenden Beispielen darstellen, in welcher Form **Do...LOOP** und **IF...ELSE...THEN** in einem Programm verwendet werden können:

1. Sie können völlig getrennt voneinander laufen

   entweder
```
DO
LOOP
IF
ELSE
THEN
```

   oder
```
IF
ELSE
THEN
DO
LOOP
```

2. **IF...ELSE...THEN** eingeschlossen in **DO...LOOP**
```
DO
  IF
  ELSE
  THEN
LOOP
```

3. **DO...LOOP** eingeschlossen in **IF...ELSE...THEN**

   entweder
```
IF
  DO
  LOOP
ELSE
THEN
```

   oder
```
IF
ELSE
  DO
  LOOP
THEN
```

IF...ELSE...THEN hat zwei Bereiche. **DO...LOOP** kann immer nur in einem Bereich vorkommen. Es ist also verboten zu schreiben

```
IF
  DO
ELSE   <-------- das ist falsch!
  LOOP
THEN
```

Wenn Sie nicht sicher sind, ob Sie sich bei der Definition eines Wortes an die Spielregeln gehalten haben oder nicht: Probieren Sie das Wort aus und sehen Sie, was passiert. Der Computer meldet ERROR 5 - und vergißt das Wort einfach.

Jetzt folgen zwei Wörter, mit denen Sie einen Programmablauf vorzeitig verlassen können: **LEAVE**, um aus **DO...LOOP** herauszukommen, und **EXIT** um gleich ein ganzes Wort zu verlassen.

LEAVE darf nur innerhalb **DO...LOOP** (oder **DO...+LOOP**) verwendet werden.Es geht nicht direkt heraus, sondern setzt den Schleifenzähler auf den Endwert, so daß **LOOP** beim nächsten Durchgang die Schleife beenden muß.

52

EXIT kann überall eingesetzt werden, außer in DO...LOOP (oder DO... +LOOP). Gleichgültig, welche Bedeutung das Wort hat, in dem EXIT vorkommt, der Computer verläßt es sofort und geht weiter zum nächsten Wort.

Schließlich folgen noch zwei Wörter, die sehr nah mit I, I' und J verwandt sind. Um ihre Bedeutung zu verstehen, müssen Sie wissen, daß es im Computer in Wirklichkeit zwei stacks gibt: Den einen, den Sie längst kennen- und liebengelernt haben (der DATA-stack), und einen zweiten, welcher RETURN-stack heißt. Wird ein FORTH-Wort ausgeführt, muß der Computer wissen, wohin er nach der Ausführung zurückkehren soll (=RETURN). Dies gelingt ihm mit Hilfe der Rücksprung-Adresse (return-adress), die im RETURN-stack gespeichert ist.

Benutzt nun ein Wort ein zweites, dieses ein drittes und das Dritte wiederum ein viertes, dann stapelt der Computer fein säuberlich alle Adressen übereinander und findet wieder seinen Weg zurück zum ersten Wort.

Dieser RETURN-stack ist aber nicht ausschließlich für den interen Gebrauch des Computers bestimmt, sondern Sie können ihn auch selbst benutzen, um vorübergehend Zahlen aus dem DATA-stack abzulegen.

Dazu brauchen Sie die Wörter >R und R>.

>R (Zahl im DATA-stack-) wird gesprochen "nach R" und überträgt eine Zahl von der Spitze des DATA-stack an die Spitze des RETURN-stack.

R>(- Zahl aus dem RETURN-stack) heißt "R von" und ist die Umkehrung von >R. Es überträgt eine Zahl aus dem RETURN-stack in den DATAstack.

Da der RETURN-stack normalerweise für Rücksprung-Adressen verwendet wird, muß die Anzahl der verwendeten >R und R> innerhalb einer Wortdefinition ausgeglichen sein. Sie können zwischen >R und R> auch nicht EXIT verwenden, da dann im RETURN-stack keine gültige Adresse steht.

Was haben nun diese beiden mit den Wörtern I, I' und J zu tun? DO-Schleifen legen ihren Zähler und ihren Grenzwert ebenfalls im RETURN-stack ab (wobei der Zähler ganz oben ist). I, I' und J machen nun nichts anderes, als eine Kopie des jeweiligen Platzes vom RETURN-stack im DATA-stack abzulegen: I kopiert den obersten Wert, I' den zweiten, und J den dritten von oben. Sie können also damit die Zahlen wieder zurückkopieren, die Sie mit >R auf dem RETURN-stack abgelegt haben. Es bedeutet aber weiterhin, daß >R und R> auch innerhalb einer DO-Schleife ausgeglichen sein müssen.

Zusammenfassung:

BEGIN.....UNTIL

BEGIN.....WHILE.....REPEAT

DO.....LOOP

DO.....+LOOP

LEAVE zum vorzeitigen Abschluß einer DO...LOOP-Schleife

EXIT zum Verlassen eines Wortes

I, I', J, >R, R>

 53

Übungen:

1. Mit dem folgenden Wort **PRIM** kann man prüfen, ob eine Zahl Primzahl ist oder nicht (eine Primzahl läßt sich ohne Rest nur durch 1 und sich selbst teilen. 2, 3 und sind sind Primzahlen, 4 ist keine denn sie ist ohne Rest durch 2 teilbar).

   PRIM läßt die zu prüfende Zahl, den Operanden, auf dem stack zurück. Davor steht noch eine andere Zahl, die den Wert Ø hat, wenn der Operand eine Primzahl war.

```
: PRIM
  ( Zahl - Zahl, Ø für Primzahl)
   2
  BEGIN
    ( Zahl, Zahl durch die geteilt wird)
    OVER OVER DUP * < Ø=
  WHILE
    OVER OVER MOD Ø=
    IF
      EXIT
    THEN
    1+
  REPEAT
  DROP Ø
;
```

Bei **BEGIN** enthält der stack die Zahl selbst an zweiter Stelle und darüber einen Probier-Divisor, durch den sie geteilt wird. Der Divisor beginnt mit 2 und wird in jedem Durchlauf um 1 erhöht. Es genügt aber, diesen Vorgang nur bis zur Quadratwurzel der untersuchten Zahl fortzusetzen (Warum?)

Auf diese Weise läuft das Programm ab, solange das Quadrat des Probier-Divisors noch kleiner ist als die Zahl selbst. Sind alle möglichen Divisoren ausprobiert worden und keiner hat die Bedingung erfüllt, entfernen wir den letzten und stellen die Ø in den stack, da wir jetzt wissen, daß die Zahl eine Primzahl ist.

Läßt sich andererseits die Zahl teilen, verlassen wir das Programm direkt mit **EXIT**. Im stack bleibt die Zahl und der letzte Divisor zurück.

Entwerfen Sie ein Wort **PRIM?**, das nach der Prüfung im stack eine 1 (wahr) für eine gefundene Primzahl, und eine Ø (falsch) für eine dividierbare Zahl abstellt. (Hinweis: **PRIM** sollten Sie natürlich dafür verwenden).

Hier haben Sie ein Wort **PRIMES**, das alle Primzahlen bis zu einem einzugebenden Grenzwert ermittelt und anzeigt. Es verwendet Ihr **PRIM?**

```
: PRIMES
  ( Grenzwert - )
  1
  DO
    I PRIM?
    IF
      I .
    THEN
  LOOP
;
```

2. Eine Zahl potenzieren ("zur Potenz erheben") bedeutet, sie mehrmals mit sich selbst zu multiplizieren. Wie oft das geschehen soll, wird durch die Potenz angegeben. Soll beispielsweise eine Zahl zur 2.Potenz erhoben werden, heißt das, daß sie mit sich selbst zu multiplizieren ist:

   6 zur 2. Potenz = 6 * 6 = 36

   6 zur 3. Potenz = 6 * 6 * 6 = 216

   Normalerweise wird dieser Sachverhalt in der Form

   6^3 ← Exponent
   Basis

   geschrieben, eine weit verbreitete Schreibweise ist aber auch

   6 ↑ 3.

   Entwickeln Sie ein FORTH-Wort ↑ , das eine Zahl potenziert (Zahl,Potenz - Ergebnis).

   Was tut Ihr ↑ , wenn die Potenz 1 oder Ø ist? Eine Zahl zur 1. Potenz ist die Zahl selbst, und es gibt gute mathematische Gründe zu sagen, daß eine Zahl zur Potenz Ø = 1 ist. Bringen Sie Ihr ↑ dazu, alles genau zu tun, und vergleichen Sie es dann mit unserem:

```
: ↑
  ( Zahl,Potenz - Zahl zur Potenz erhoben)
  1 SWAP ?DUP
  IF
    Ø
    DO
      OVER *
    LOOP
  THEN
  SWAP DROP
;
```

   Beachten Sie, wie mit ?DUP, IF und THEN die DO...LOOP-Schleife übergangen wird, wenn Sie sie nicht brauchen.

3. Mit dem folgenden unwahrscheinlich nützlichen Wort können Sie Ihren Computer auf ewige Zeiten beschäftigen (es sei denn, Sie unterbrechen ihn mit BREAK).
   Verwenden Sie dafür BEGIN und Ø UNTIL.

```
: OED
  ." ER LAEUFT"
  BEGIN
  ." UND LAEUFT"
  Ø
  UNTIL
;
```

 55

KAPITEL 11

TÖNE UND MUSIK

Der Jupiter ACE hat einen eingebauten Lautsprecher, mit dem Sie Ihre Wörter bei Bedarf durch Töne oder Geräusche untermalen können. Das Wort, das Sie dazu brauchen, ist **BEEP**. Es erwartet zwei Zahlen auf dem stack. D e oberste Zahl definiert die Länge des Tons in Millisekunden, die zweite bestimmt die Tonhöhe. (Technisch ausgedrückt handelt es sich dabei um die Dauer einer Schwingung in Einheiten von 8 µs).

Sie werden es vermutlich vorziehen, die folgende Tabelle zu benutzen, die die Tonhöhen für 7 Oktaven in Halbtonschritten darstellt:

C	1911	956	478	239	119	6Ø	3Ø
B	2Ø25	1Ø12	5Ø6	253	127	63	32
$B^bA^\#$	2145	1Ø73	536	268	134	67	34
A	2273	1136	568	284	142	71	36
$A^bG^\#$	24Ø8	12Ø4	6Ø2	3Ø1	15Ø	75	38
G	2551	1276	638	319	159	8Ø	4Ø
$F^\#G^b$	27Ø3	1351	676	338	169	84	42
F	2863	1432	716	358	179	89	45
E	3Ø34	1517	758	379	19Ø	95	47
$E^bD^\#$	3214	16Ø7	8Ø4	4Ø2	2Ø1	1ØØ	5Ø
D	34Ø5	17Ø3	851	426	213	1Ø6	53
$C^\#D^b$	36Ø8	18Ø4	9Ø2	451	225	113	56
C	3822	1911	956	478	239	119	6Ø

Als Faustregel läßt sich sagen: Je kleiner die Zahl, desto höher der Ton.

Wollen Sie aber Musik machen, dann brauchen Sie ein wenig mehr Aufwand. Verfolgen Sie das an dem Lied " Alle Vögel sind schon da".

Für jede Note müssen Tonhöhe und Dauer festgelegt werden. Statt mit Millisekunden zu arbeiten, ist es besser, die Dauer der kürzesten Note einmal als Variable zu definieren und alle anderen Tonlängen als Vielfache davon zu bestimmen.

Die kürzeste Tondauer in unserem Lied ist eine Achtel-Note. Wir definieren:

```
2ØØ VARIABLE ACHTEL

: N
  ( Tonhöhen-Zahl, Länge in Achteln)
  ACHTEL @ * BEEP
;
```

56

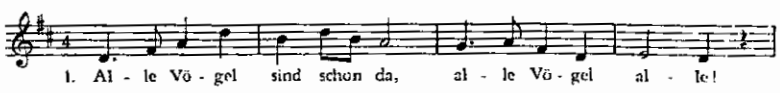
Das "kleine c" als Viertelnote mit Punkt (drei Achtel lang) ist jetzt also

```
478 3 N
```

Der Name **N** wurde bewußt kurz gewählt, da er ja oft eingegeben werden muß. Durch die Verwendung von **ACHTEL** haben wir den zusätzlichen Vorteil, daß die ganze Melodie schneller oder langsamer gespielt werden kann, einfach indem wir den Wert von **ACHTEL** ändern. Der Wert **200** für die Variable **ACHTEL** ist schon sehr lang.

In unserem Lied kommen mehrere Wiederholungen vor. Es besteht aus zwei gleichen Elementen

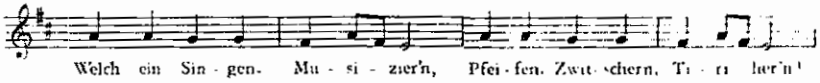
Teil 1



wird definiert durch

```
: TEIL1
  ( alle Vögel sind schon da)

  478 6 N 379 2 N 319 4 N
  239 4 N 284 4 N 239 2 N
  284 2 N 319 8 N 358 6 N
  319 2 N 379 4 N 478 4 N
  426 8 N 478 8 N

;
```

Teil 2



wird definiert durch

```
: TEIL2
  ( Singen,Pfeifen...)

  319 4 N 319 4 N 358 4 N
  358 4 N 379 4 N 319 2 N
  379 2 N 426 8 N

;
```

Stellen wir nun das ganze Lied zusammen

```
: ALLE
  TEIL1 TEIL2
  TEIL2 TEIL1
;
```

<u>Zusammenfassung:</u>

FORTH-Wort **BEEP**

57

Übungen:

1. Anstatt die Tonhöhe aus der Tabelle zu bestimmen, kann man sie auch in Beziehung zu einer Art Standard-Note setzen und sie berechnen. Dazu muß man berücksichtigen, daß die Noten in einem logarithmischen Verhältnis zueinander stehen.

   Um also eine Note um eine Oktave zu erhöhen, muß die Tonhöhen-Zahl mit 1/2 multipliziert werden. Sie können dies aus der Tabelle deutlich erkennen. Für die Erhöhung um einen Halbton muß die Tonhöhe mit der zwölften Wurzel aus 1/2 mulitipliziert werden (eine Oktave umfaßt 12 Halbtöne). Das ist die Zahl Ø.94387431 oder näherungsweise 17843/189Ø4.

   Damit kommen wir zu einer Methode zur Berechnung von Tonhöhen-Zahlen aus entsprechenden Halbtönen:

   a) Beginnen Sie mit einer ziemlich tiefen Note mit bekannter Höhenzahl, z.B. 3822 für das tiefste C in der Tabelle

   b) Ermitteln Sie die Halbtöne in Form ganzer Oktaven plus "überhängender" Halbtöne. (Die Note "G" mit der Höhenzahl 8Ø liegt zum Beispiel um 2 Oktaven und 7 Halbtöne über dem kleinen "c").

   c) Multiplizieren Sie die Basiszahl für jeden Halbton mit 17843/189Ø4 und dividieren Sie das Ergebnis für jede Oktave durch 2. Damit erhalten Sie die gewünschte Tonhöhen-Zahl.

   Im folgenden Wort **SEMI** ist diese Methode realisiert. Sie nimmt aus dem stack die Tonhöhe in Halbtönen über dem kleinen c, und stellt die Höhen-Zahl für **BEEP** zurück.

```
: SEMI
  ( Halbtöne über kleinem c - Tonhöhe)
  36 + ( Halbtöne über dem tiefen C)
  12 /MOD SWAP ( Zahl Oktaven, Zahl Halbtöne)
  3822 SWAP ?DUP
  IF
    ( Multiplikation mit 17843/189Ø4 f.jeden Halbton)
    Ø
    DO
     17843 189Ø4 */
    LOOP
  THEN
  SWAP ?DUP
  IF
   ( Division durch 2 für jede Oktave)
   Ø
   DO
    2 /
   LOOP
  THEN
;
```

Sie können nun **SEMI** mit **BEEP** testen. Wegen der vielen Rechenarbeit entsteht vor den höheren Noten innerhalb einer Oktave eine deutlich längere Pause als etwa vor C. In Kapitel 2Ø werden wir eine Methode kennenlernen, wie wir Potenzen von 17843/189Ø4 getrennt speichern können, so daß für die überhängenden Halbtöne nur eine Multiplikation und eine Division erforderlich wird.

58

2. Versuchen Sie **BEEP** mit kleinem Operanden. Der kleinste Tonhöhenwert, der noch einen hörbaren Ton erzeugt, ist 7. Bei noch kleineren Werten spielt der Computer nicht mehr mit und erzeugt ein endloses Knackgeräusch.

   Die kürzeste Notendauer, die Sie benutzen können, hängt von der Tonhöhe ab: Ist die Länge geringer als 1/125 des Höhenwerts, dann erzeugt der Computer einen Dauerton in richtiger Höhe. Bevor dieser Fall eintritt, müssen die Noten aber schon extrem kurz sein.

 59

KAPITEL 12

DER ZEICHENSATZ

Bisher hatten wir es nur mit Buchstaben, Ziffern, Satz- und einigen Sonderzeichen zu tun, die auch der Computer kennt. Jedes von ihnen wird im ACE durch eine Zahl zwischen Ø und 255 dargestellt. Man nennt diese Zahl den ASCII Code eines Zeichens (ASCII bedeutet "American Standard Code for Information Interchange", auf Deutsch: Amerikanischer Standardcode für Informationsaustausch), und der Computer erkennt alle Zeichen nur in Form ihrer Codes.

Wollen Sie den ganzen Zeichenvorrat sehen, dann definieren Sie ein Wort

```
: CHAR
  ( zeigt den Zeichensatz an)
  256 Ø
  DO
    I EMIT
  LOOP
;
```

EMIT nimmt einen ASCII Code aus dem stack und stellt das dazugehörige Zeichen auf dem Bildschirm dar. Sie können sofort zwei Gruppen unterscheiden: die Zeichen mit den Codes Ø bis 128 sind weiß auf schwarz, und die mit 128 bis 255 sind schwarz auf weiß (die sogenannte "Invers Video"-Darstellung). Ansonsten sind sie gleich.

Der Buchstabe A hat in weiß-auf-schwarz-Darstellung den ASCII Code 65, und in schwarz-auf-weiß-Darstellung 65 + 128 = 193.

Die Zeichen mit den ASCII Codes 32 bis 127 sind auf der ganzen Welt gleich (eben standardisiert); wobei es nur geringfügige Unterschiede bei Sonderzeichen gibt, die nicht in jedem Land verwendet werden (z.B. das Zeichen £)

Die Zeichen mit Code Ø bis 31 sind nicht einheitlich festgelegt. In ASCII werden sie als Steuerzeichen eingesetzt, die zwar nicht angezeigt werden können, aber bestimmte Funktionen erfüllen. Der ACE benutzt beispielsweise ASCII 13 als "Rücklauf", die MARKE geht von einer Zeile zum linken Rand der Folgezeile. Alle übrigen werden als "Graphic-Zeichen" verwendet, die Ihnen für spezielle Anwendungen zur Verfügung stehen.

Die Zeichen sind Muster aus schwarzen und weißen Quadraten. Die Fläche, in der ein normales Zeichen dargestellt werden kann, ist in 4 kleinere Quadrate aufgeteilt:

Da jedes der kleineren Quadrate schwarz oder weiß sein kann, gibt es 2*2*2*2 = 16 Möglichkeiten, die im ACE mit den Codes 16 bis 23 und 144 bis 151 festgelegt sind.

Diese Zeichen können mit den Graphics Modus direkt über die Tastatur eingegeben werden. Durch Drücken der Tasten SHIFT und 9 (dort steht auch GRAPHICS) wird die MARKE in G geändert und die Zifferntasten erzeugen die auf ihnen dargestellten Symbole. Die anderen 8 Symbole erhalten Sie, nachdem Sie zusätzlich SHIFT und 4 (INVERSE VIDEO) gedrückt haben.

60

Zeichen	Code	Zeichen	Code
	16		144
	17		145
	18		146
	19		147
	20		148
	21		149
	22		150
	23		151

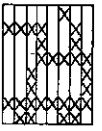
Sie werden feststellen, daß auch die anderen Tasten Grafik-Zeichen erzeugen. Sie haben zwar das gleiche Muster, aber natürlich andere ASCII Codes. Es gibt vier Gruppen von Code-Nummern zur Darstellung der ersten acht nicht inversen Zeichen: 0 bis 7, 8 bis 15, 16 bis 23 und 24 bis 31. Da Sie jedes Zeichen frei definieren können, verwenden Sie die Zeichen 0 bis 15 und 24 bis 31 für Ihre eigenen Kreationen.

Sie können selbst errechnen, welche Taste welches Zeichen erzeugen kann:

1. Stellen Sie den ASCII Code der Taste fest (s.Anhang A)
2. Dividieren Sie den Code durch 32
3. Der Rest ist der Code des Zeichens, das dargestellt wird
4. Sie kommen durch Addition von 128 in den Inverse-Video-Modus

Der Buchstabe a hat den ASCII Code 97. 97 : 32 = 3, Rest 1. Also: Taste a hat das gleiche Grafik-Symbol wie ASCII Code 1.
Merken Sie sich einfach, daß a oder A im Graphik-Modus die 1, b oder B die 2 ergibt usw. bis zu z oder Z = 26.

Sie können Zeichen selbst definieren. Nehmen wir an, Sie wollen das Bild einer kleinen Lokomotive für irgendein Spiel zeichnen. Alle Zeichen benutzen ein Raster von 8 x 8 Punkten (schauen Sie einmal genau auf den Bildschirm), also müssen Sie zuerst das Bild der Lok in diesem Raster entwerfen.



Jedes Kreuz bedeutet einen weißen Punkt auf dem Bildschirm, eine Leerstelle dagegen einen schwarzen Punkt. Jetzt ist festzulegen, welchen ASCII Code die Maschine haben soll. Nehmen wir 1, das entspricht dem A im Graphik-Modus. Jeder andere Code geht natürlich auch (aber Vorsicht bei Codes über 128, sie sind in invers). Es ist jedoch besser, die sonst nicht verwendeten Codes ASCII Ø bis 15 oder 24 bis 31 zu nehmen.

Definieren Sie folgendes Wort

```
: GR
  8 * 11263 + DUP
  8 +
  DO
   I C! -1
  +LOOP
;
```

GR erwartet 9 Zahlen im stack. Die oberste ist der ASCII Code des Zeichens, das wir ändern wollen, z.B. 1, die übrigen Zahlen beschreiben die acht Punktreihen. Dabei ist die höchste Reihe ganz unten und die tiefste Reihe ganz oben im stack. Geben Sie jetzt bitte ein:

```
2 BASE C!

ØØØØØ1ØØ
1111ØØ1Ø
ØØØ1ØØ1Ø
ØØØ11111
ØØ1ØØØØ1
ØØ1ØØØØ1
11111111
Ø11ØØ11Ø

DECIMAL
```

(die genaue Bedeutung von **BASE** und **DECIMAL** wird in Kapitel 16 erläutert. Wir haben die sogenannte Binärschreibweise benutzt).

Wie Sie erkennen können, ist aus jedem x im Entwurf eine 1, und aus jeder Leerstelle eine Ø geworden. Jetzt müssen Sie nur noch 1 (für den ASCII Code) und **GR** eingeben, und das Zeichen mit Code 1 ist jetzt eine Lok. Gehen Sie mit SHIFT 9 in den Graphik-Modus und drücken Sie A. Mit Invers Video (SHIFT 4) und danach A erscheint sie schwarz auf weißem Hintergrund. Sie können das Maschinchen jetzt wie jedes andere Zeichen einsetzen, indem Sie es nach ." bringen oder mit 1 EMIT abrufen. Sie können es sogar als eigenes Wort benutzen, z.B.

```
:  [illegible]
  ." Auf der schwäb'schen Eisenbahn"
;
```

Wie arbeitet nun GR? Stellen Sie sich eine Reihe mit 8 Punkten vor. Jeder Punkt kann schwarz oder weiß sein, es gibt also 2^8 = 256 verschiedene Kombinationsmöglichkeiten in einer Reihe. Jede Reihe kann als ein Byte codiert werden. Insgesamt haben wir 8 solcher Reihen, brauchen also 8 Bytes, die wir im Speicher ablegen.

Die Speicheradressen von 11264 bis 12287 werden für die Speicherung der ASCII Zeichencodes Ø bis 127 verwendet. Sie sind nach Code-Nummern gespeichert. Die oberste Reihe hat die Adresse 11264+8* ASCII Code, und die achte Reihe ist sieben Plätze weiter hinten. Das sieht dann so aus:

 62

Adresse	Reihe, die dort gespeichert ist
11264	oberste Reihe für ASCII Ø
11265	2. Reihe für ASCII Ø
11266	3. Reihe für ASCII Ø
11267	4. Reihe für ASCII Ø
11268	5. Reihe für ASCII Ø
11269	6. Reihe für ASCII Ø
11270	7. Reihe für ASCII Ø
11271	unterste Reihe für ASCII Ø
11272	oberste Reihe für ASCII 1
11273	2. Reihe für ASCII 1
11274	3. Reihe für ASCII 1
..	..
..	..
..	..
..	..
11279	unterste Reihe für ASCII 1
..	..
..	..

Wollen wir ein eigenes Zeichen für ASCII 1 entwickeln, müssen die entsprechenden Zahlen in die Speicherplätze 11272 bis 11279 gebracht werden. **GR** berechnet diese Adressen und belegt die Plätze von 11279 an rückwärts.

Dazu wird ein neues Wort **C!** benutzt.

C! speichert nur ein Byte weg (Byte, Adresse -) im Gegensatz zu **!**, das bekanntlich zwei benachbarte Adressen belegt.

Ein entsprechendes Wort für das Lesen nur eines Speicherplatzes ist **C@** . (Adresse - Byte). Es entspricht dem Zeichen **@**.

FORTH hat für Behandlung von Zahlen immer zwei einander entsprechende Wörter. Einige benutzen ganze Zahlen aus 2 Bytes (wie z.B. **@** oder **!**), andere nur ein Byte (wie **C@** oder **C!**). Da der ASCII Code eines Zeichens (=Character) ein Byte benutzt, beginnen diese Wörter meist mit **C**

Beachten Sie bitte: Der Speicherbereich, der die Punkt-Muster enthält, kann nur beschrieben, aber nicht in den stack zurückgelesen werden. Nur die Anzeige auf dem Bildschirm ist möglich.

Für die Verarbeitung von Zeichen gibt es ein besonderes Wort: **ASCII**. Es erspart Ihnen, jedesmal in die Tabelle zu schauen, wenn Sie den ASCII Code eines Zeichens wissen wollen. **ASCII** nimmt das erste Zeichen des unmittelbar folgenden Wortes und legt dessen ASCII Code im stack ab.

ASCII ANTON

druckt z.B. 65 aus. ANTON ist kein definiertes Wort, aber das ist **ASCII** gleichgültig. Es nimmt einfach das nächste Zeichen nach einem Zwischenraum. Deshalb kann es übrigens auch den Zwischenraum selbst nicht umschlüsseln.

Weitere Wörter für die Verarbeitung von Zeichen sind:

SPACE, SPACES, CLS, AT und **TYPE**

 63

SPACE (-) gibt auf dem Bildschirm einen Zwischenraum aus

SPACES (n -) nimmt die oberste Zahl vom stack und erzeugt ebenso viele Zwischenräume, wenn die Zahl positiv ist

CLS (-) löscht den gesamten Bildschirminhalt. Es löscht auch den Eingabebereich bis auf eine einzige Zeile. Wollen Sie **CLS** über die Eingabe benutzen, dürfen nicht zu viele Wörter danach folgen, weil sie sonst verlorengehen.

AT (Zeile,Spalte -) nimmt zwei Zahlen aus dem stack und verwendet sie als Koordinaten für die MARKE. Die zweite Zahl von oben definiert eine Zeile im Bildschirm (Zeilenzählung von Ø bis 22, Ø ist die oberste Zeile). Die oberste Zahl definiert eine Spalte (Spaltenzählung beginnt mit Ø am linken Rand und geht bis 31). **AT** definiert auf diese Weise Zeile und Spalte, in der die nächste Bildschirmanzeige beginnen soll.

TYPE (Adresse, Anzahl von Zeichen -) zeigt auf dem Bildschirm Zeichen aus dem Speicher an. Es beginnt bei einer im stack abgelegten Adresse und zeigt die im nächsten Platz des stack angegebene Zahl von Zeichen an.

Zusammenfassung:

Zeichensatz und ASCII Codes

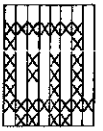
Entwurf neuer Zeichen

FORTH-Wörter: **C!**, **C@**, **EMIT**, **ASCII**, **SPACE**, **SPACES**, **CLS**, **AT**, **TYPE**

64

Übungen:

Definieren Sie ein Zeichen mit ASCII Code 2 als Eisenbahnwaggon:



und geben Sie die folgende Wortdefinition **GO** ein. Beim Eintasten müssen Sie zwei Zwischenräume vor dem Zug eingeben.

```
: GO
  ( Länge des Pfeiftons - Länge des Pfeiftons)
  BEGIN
    CLS 22 Ø
    DO
     32 Ø
     DO
      J I AT ."          "
      DUP 2ØØ SWAP BEEP ( Pfiff)
     LOOP
    LOOP
    Ø
  UNTIL
;
```

GO braucht eine Zahl im stack, die bestimmt, wie schnell der Zug fährt.

2. Definieren Sie ein Wort

```
: CODEA
  ASCII abcde .
;
```

Zeigen Sie das Wort mit **LIST** wieder an. Die Buchstaben bcde sind weggefallen, da sie keine Rolle spielen.

Wir haben vorhin behauptet, daß **ASCII** ein Wort aus der Eingabezeile nimmt und den ASCII Code des ersten Zeichens in den stack stellt.

Das geschieht aber nicht, wenn wir **CODEA** laufen lassen. Es nimmt vielmehr einen Wert von dem Wort, das während der Definition von **CODEA** in der Eingabezeile war. Es verhält sich innerhalb einer Wortdefinition also anders als im Eingabebereich. Dieser Unterschied ist beabsichtigt.

3. Entwerfen Sie Schachfiguren, Spielkarten-Symbole oder was Ihnen sonst gerade einfällt.

4. Wenn Sie BASIC kennen, wissen Sie, was TAB ist. Im FORTH gibt es kein entsprechendes Wort. Es ist aber einfach zu definieren:

```
: TAB
  ( Tabstop -)
  15388 @ - 31 AND SPACES
;
```

(wie das Wort funktioniert, ist im Augenblick noch nicht so wichtig für Sie).

TAB nimmt eine Zahl aus dem stack und verwendet es als Spalten-Nummer zwischen Ø und 31 (ist die Zahl größer, wird sie durch 32 dividiert und der Rest verwendet). **TAB** stellt dann soviel Zwischenraum bereit, daß die nächste anzuzeigende Zahl in die Spalte paßt (wenn nicht in der gleichen, dann in der folgenden Zeile).

Im folgenden Wort **TABELLE** wird dies mit einer 4-spaltigen Tabelle gezeigt:

```
: TABELLE
  ( Zähler bis -)
  Ø
  DO
   I 8 * TAB
   I .
  LOOP
  CR
;
```

5. Experimentieren Sie mit **TYPE**.

Ø 1ØØ **TYPE** druckt wirres Zeug, weil die Bytes ab Ø keine sinnvollen Zeichen sind. Es sind vielmehr fest eingebaute Befehlscodes.

Interessanter ist das schon 8192 5ØØ **TYPE**, weil 8192 der Anfang des Speicherbereichs für Bildschirminformationen ist. **TYPE** liest also vom Bildschirm und schreibt gleich wieder zurück.

Sie können sich ein **TYPE** unter Verwendung von **C@**, **EMIT** und **DO** Schleife selbst entwickeln.

6. Zum Schluß ein kleines lustiges Spiel. Dazu müssen Sie zunächst einige Symbole mit **GR** definieren. Wir haben Ihnen die entsprechenden Codes gleich in Dezimalschreibweise vorgegeben.

Geben Sie zunächst die Definition von **GR** von Seite 61 ein, danach geben Sie ein:

Ø	Ø	7	15	15	31	18	5Ø	1	GR
Ø	Ø	Ø	128	128	192	64	96	2	GR
63	48	112	127	127	32	16	8	3	GR
224	96	112	24Ø	24Ø	32	64	128	4	GR
63	48	112	127	127	16	32	64	5	GR
224	96	112	24Ø	24Ø	64	32	16	6	GR
Ø	126	2	2	62	2	126	Ø	7	GR

Damit haben wir ein kleines Monster definiert, das wir jetzt über den Bildschirm laufen und ein bißchen arbeiten lassen wollen.

66

Wir definieren zunächst das Wort für die Bewegung. Beachten Sie dabei aber bei den Symbolen in ." " die vorgesehenen Leerschritte, damit Ihr Männchen seine Arbeit einwandfrei verrichten kann.

```
: GEH
  DUP DUP 2 MOD
  IF
    11 SWAP AT ." GABG " ( AB im GRAPHICS-Modus)
    12 SWAP AT ." GCDG " ( CD im GRAPHICS-Modus)
    1ØØ
  ELSE
    11 SWAP AT ." GABG " ( AB im GRAPHICS)
    12 SWAP AT ." GEFG " ( EF im GRAPHICS)
    2ØØ
  THEN
  1ØØ BEEP
;
```

Im Listing können Sie das Männchen schon erkennen. Mit **MOD** wird entschieden, ob es die Beine spreizt oder zusammenklappt.

```
: FEHL
  CLS 1Ø 1 AT ." Jupiter ACGGG" ( G in Graphics)
  9 1ØØØ BEEP 18 1

  DO
    I GEH ( Schleife 1)
  LOOP
  5ØØ 1ØØØ BEEP 8 17

  DO
    I GEH -1 ( Schleife 2)
  +LOOP
  1Ø GEH 29 11

  DO
    1Ø I AT ." GGG " ( G in Graphics) ( Schleife 3)
    I GEH 6ØØ 2ØØ BEEP
  LOOP
  1Ø 28

  DO
    1Ø I AT ." E " ( Schleife 4)
    I 1- GEH
    6ØØ 2ØØ BEEP -1
  +LOOP
  29 11

  DO
    I GEH ( Schleife 5)
  LOOP
;
```

Unser Männchen geht unter dem Schriftzug entlang (Schleife 1), stutzt, als es das falsche E erkennt und läuft zurück (Schleife 2). Es schleppt das falsche E weg (Schleife 3) und bringt das richtige (Schleife 4). Unter der Last geht es natürlich langsamer. Das erreichen wir mit dem längeren **BEEP**. Zum Schluß verläßt es (in Schleife 5) den Platz.

KAPITEL 13

ZEICHNEN

Mit den kleinen Quadraten aus Kapitel 12 können Sie den Bildschirm mit beliebigen Mustern aus schwarzen und weißen Kästchen verzieren. Es ist nur eine etwas mühselige Arbeit. Einfacher geht das Zeichnen dagegen mit dem Wort **PLOT**, mit dem Sie einen Punkt an eine beliebige Stelle im Bildschirm setzen können.

Jede dieser Quadrat-Positionen nennt man ein Pixel (engl.: picture element). Der Bildschirm hat 64 Pixels in der Breite und 48 in der Höhe. Sie können in der Höhe aber nur 46 Pixels benutzen, da die unterste Zeile als Eingabezeile reserviert bleibt.

Um ein Pixel zu definieren, brauchen Sie zwei Zahlen, seine Koordinaten. Die erste ist die X-Koordinate von links nach rechts. Ganz links steht das Pixel mit der Koordinate Ø, ganz rechts das mit der Koordinate 63.
Die zweite Zahl, die Y-Koordinate, verläuft von unten nach oben. Das unterste Pixel hat Ø, das oberste 45.

Im folgenden Diagramm ist das Koordinatensystem noch einmal dargestellt (s. Seite 72).

Sind die Koordinaten festgelegt, müssen Sie dem ACE noch sagen, was mit dem Pixel geschehen soll. Sie haben dafür vier Möglichkeiten (man sagt dazu "Plotting Modus"):

Pixel auf schwarz setzen (unplot)	= Ø
Pixel auf weiß setzen (plot)	= 1
Pixel nicht verändern (move)	= 2
Pixel in Gegenfarbe ändern (Change)	= 3

Die Zahlen nach dem Gleichheitszeichen definieren den entsprechenden Code.

PLOT braucht also 3 Zahlen im stack (X-Koordinate, Y-Koordinate, Plotting Modus -).

Geben Sie ein

```
30 20 ! PLOT
```

und Sie erhalten ein kleines weißes Quadrat etwa in der Mitte des Bildschirms.

Jetzt können Sie fleißig üben und Punkte anzeigen lassen. Eines wird Sie aber stören: immer wenn der ACE einen Befehl ausgeführt hat, schreibt er noch einmal an, was er getan hat und setzt OK dahinter. Das könnte Ihr Kunstwerk etwas beeinträchtigen. Deshalb gibt es ein Wort, mit dem man diese Meldungen unterdrücken kann. Es heißt **INVIS** (invisible = unsichtbar). Umgekehrt werden die Meldungen wieder sichtbar, wenn Sie **VIS** verwenden.

Geben Sie ein

```
INVIS CLS
```

und malen Sie jetzt nach Herzenslust.

Zusammenfassung:

Pixels, X- und Y-Koordinaten
FORTH-Wörter **PLOT**, **INVIS**, **VIS**

68

Übungen:

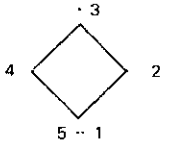
1. Mit einem Wort **DRAW** (=zeichnen) können Sie recht gut gerade Linien zeichnen. Es hat drei Operanden

   (x, y, Plotting Modus -)

   Der Anfangspunkt der Linie ist das letzte mit **PLOT** oder **DRAW** ausgegebene Pixel, und der Endpunkt liegt x- Pixels darüber und y-Pixels rechts davon. x und y haben also eine gewisse Ähnlichkeit mit den x- und y-Koordinaten eines Pixels in **PLOT**, werden jedoch vom Anfangspunkt der Linie aus gerechnet und nicht von der unteren linken Ecke aus. Sie können also auch negativ sein, wie z.B. in der Wortfolge

```
 30   5 1 PLOT
 10  10 1 DRAW
-10  10 1 DRAW
-10 -10 1 DRAW
 10 -10 1 DRAW
```

   die ein Quadrat zeichnet



   Die Zahlenfolge 1...5 gibt an, in welcher Reihenfolge die Seiten des Quadrats gezeichnet werden.

   Leider gibt es **DRAW** nicht fertig im Wörterbuch. Wir müssen es aus einigen untergeordneten Wörtern definieren:

```
: SGN
  0> DUP + 1-
;
: DRAW1
  ROT ROT OVER SGN OVER SGN
  4 ROLL ABS 4 ROLL ABS
  OVER OVER <
  ROT ROT 3 PICK
  IF
    SWAP
  THEN
;
: DIAG
  6 PICK 6 PICK
;
: GERADE
  4 PICK
  IF
    0 6 PICK
  ELSE
    6 PICK 0
  THEN
;
: SCHRITT
  15408 C@ + SWAP
  15407 C@ + SWAP
  9 PICK PLOT
;
```

Fortsetzung Folgeseite

Fortsetzung Wörterdefinition von Vorseite:

```
: DRAW
  DRAW1 2 PICK DUP 2 /
  SWAP ?DUP
  IF
    Ø
    DO
      OVER + DUP 4 PICK >
      IF
       3 PICK - DIAG
      ELSE
       GERADE
      THEN
       SCHRITT
      LOOP
  THEN
  7 Ø
  DO
    DROP
  LOOP
;
```

Hierzu gibt es einiges zu erklären:
Zunächst soll m den größeren und n den kleineren der Werte |x| bezw. |y| darstellen. |x| und |y| sind die Absolutwerte von x und y, d.h. also Werte ohne Vorzeichen.

Wir bauen die Linie auf, indem wir zwei Arten von Schritten verwenden: Einen Diagonalschritt, der ein Pixel sowohl nach x als auch nach y verschiebt; und einen geraden Schritt, der nur in eine Richtung nämlich vertikal oder horizontal verläuft. Wenn wir so gleichmäßig wie möglich n Diagonalschritte und m-n gerade Schritte ausführen, dann bewegen wir uns um m Pixels in die eine und um n Pixels in die andere Richtung, also genau das, was wir wollen.

DRAW1 erarbeitet diese beiden Arten von Schritten. Es ersetzt x, y und den Plotting Modus durch die folgenden sechs Zahlen im stack (von oben nach unten):

- n, die kleinere Zahl aus |x| und |y|.
- m, die größere Zahl aus |x| und |y|.
- einen MERKER, der anzeigt, welche der beiden Zahlen die größere war: Ø, wenn |x| größer war und ein gerader Schritt horizontal geht. 1, wenn |y| größer war und ein Vertikalschritt folgt. (Wenn |x| = |y|, gibt es keinen geraden Schritt und der MERKER ist ohne Bedeutung).
- die y-Richtung eines Diagonalschrittes (1 oder -1)
- die x-Richtung eines Diagonalschrittes (1 oder -1)
- den Plotting Modus.

DIAG kopiert die beiden Richtungsangaben eines Diagonalschrittes an den TOS (sie sind 1 oder -1); **GERADE** kopiert die Richtungsangaben eines geraden Schrittes an den TOS (eine ist Ø, die andere 1 oder -1).

SCHRITT nimmt die beiden Angaben zu einem Schritt, so wie sie von **DIAG** oder **GERADE** zurückgelassen wurden, und zeichnet mit ihnen den nächsten Punkt der Linie. **SCHRITT** muß dazu wissen, wo der vorhergehende Punkt gelegen hat und benutzt dazu zwei Systemvariable, die jede ein Byte lang sind, aus den Adressen 154Ø7 und 154Ø8. (PLOT legt nämlich die beiden Koordinaten dort ab: x steht in 154Ø7, y in 154Ø8).

70

DRAW mischt jetzt die n Diagonalschritte mit den m-n geraden Schritten. Diese stehen in einem bestimmten Verhältnis zueinander, nämlich (n) : (m-n). Das Zahlenverhältnis wird in der **DO...LOOP**-Schleife simuliert. Im stack steht eine eigene Zahl zwischen 1 und m. In jedem Durchgang wird n hinzuaddiert. Ist die Summe größer als m geworden, folgt ein Diagonalschritt und m wird wieder subtrahiert. Andernfalls folgt ein gerader Schritt.

2. Experimentieren Sie mit **DRAW**, indem Sie die verschiedenen Plotting Modes einsetzen. Hierbei sehen Sie eine Anwendungsmöglichkeit für den Modus 2: Er stellt den **PLOT**-Status her, ohne die Pixels zu beeinflussen.
Der Plotting Modus 3 ist nützlicher, als Sie vielleicht annehmen, denn dank seiner besonderen Eigenschaft erhalten Sie durch zweimaliges Einsetzen von **PLOT** oder **DRAW** wieder den ursprünglichen Zustand.

Nehmen Sie z.B. (nach **INVIS, CLS**)

```
Ø 24 1 PLOT 63 Ø 1 DRAW
32 Ø 1 PLOT Ø 45 1 DRAW
```

Wollen Sie jetzt die zweite Linie wieder löschen, können Sie dies mit

```
32 Ø Ø PLOT Ø 45 Ø DRAW
```

tun. Es hinterläßt aber in der ersten Linie eine Lücke. Wiederholen Sie den Versuch durchgehend mit Modus 3. Sie haben dann eine Lücke im Kreuzungspunkt beider Linien, nach dem Löschen ist sie jedoch wieder geschlossen.

3. Zeichnen Sie auf den leeren Bildschirm

```
Ø Ø 3 PLOT 63 2Ø 3 DRAW
```

und dann versuchen Sie mit

```
-63 -2Ø 3 DRAW
```

die Linie wieder zu löschen. Es geht nicht, denn die vorhandenen Unregelmäßigkeiten führen die zweite Linie einen anderen Weg.
Um eine Linie wirklich zu treffen, müssen Sie die zweite Linie in gleicher Richtung verlaufen lassen.

4. Ein sehr nützliches Hilfsmittel für viele Anwendungen ist ein Zufallsgenerator, ein Wort, das eine zufällige Zahl erzeugt, wie etwa das Roulette. Für einen Computer ist es garnicht so leicht, Zufallszahlen zu erzeugen, da er ja festen, vorhersehbaren Befehlen folgt. Einfacher ist es, Pseudozufallszahlen zu erzeugen, Zahlen also, die einer Gesetzmäßigkeit unterliegen, die - obwohl fixiert - hinreichend komplex ist, um zufällig zu erscheinen. Hier ein Beispiel:

```
Ø VARIABLE SEED

: SEEDON
  ( - nächster Wert von SEED)
  SEED @ 75 U* 75 Ø D+
  OVER OVER U< - -
  1- DUP SEED !
;
```

Fortsetzung Folgeseite

71

```
: RND
  ( n - Pseudozufallszahl zwischen Ø und n-1)
  SEEDON U* SWAP DROP
;

: RAND
  ( Wert für SEED - )
  ?DUP Ø=
  IF
    154Ø3 @ SWAP
  THEN
  SEED !
;
```

Kümmern Sie sich (noch) nicht darum, wie das funktioniert, nehmen Sie diese Programme einfach als fertige Rezepte.

SEEDON nimmt bei jedem Durchlauf den alten Wert von **SEED**, um einen neuen zu erzeugen, den es wieder in den stack zurücklegt. Wenn Sie **SEEDON** 65536mal benutzen, ist es wieder beim Anfangswert angelangt und der Zyklus beginnt von Neuem.

RND hat einen Operanden und benutzt **SEEDON** dazu, eine Zufallszahl zu erzeugen, die kleiner ist als dieser Operand. 6 **RND** erzeugt ein Ergebnis zwischen Ø und 5, und 6 **RND** 1+ ein Resultat zwischen 1 und 6, kann also als "elektronischer Würfel" benutzt werden.

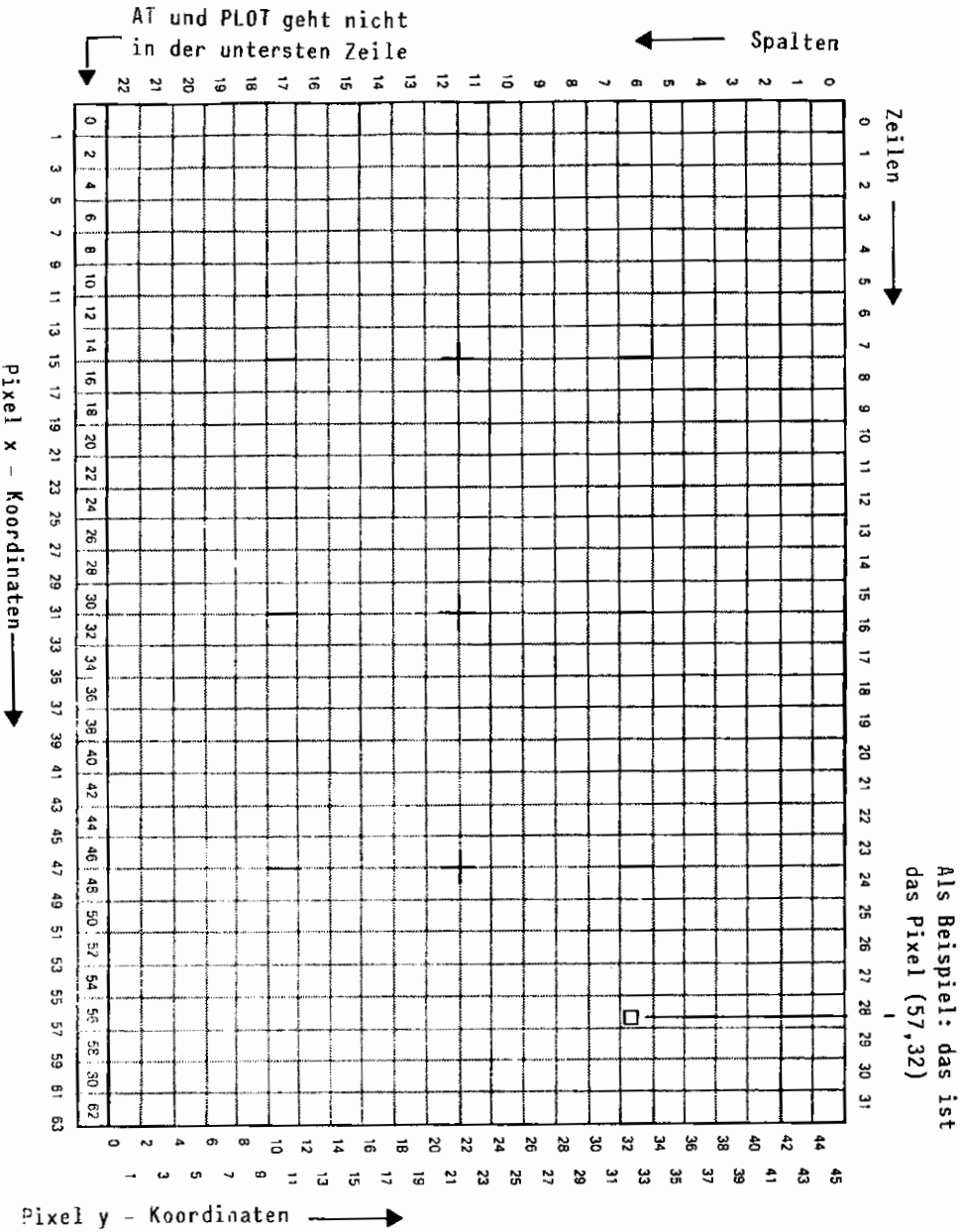
RAND startet **SEED** mit dem TOS, so daß Sie immer wissen, mit welcher Zahl die Zufallsfolge beginnt. Das kann für die Fehlersuche sehr nützlich sein, weil Sie immer wieder die gleichen Zahlen für einen Programmtest erhalten. Eine Besonderheit von **RAND** ist, daß Sie mit
Ø **RAND** als Ausgangsgröße eine Systemvariable erzeugen können, die aussagt, wieviele Fernsehbilder seit dem Einschalten des Computers abgelaufen sind (5Ø je Sekunde). Damit erhalten Sie einen noch besseren Zufallswert.

5. Tippen Sie das folgende Programm mit

   1 **MUSTER** oder 3 **MUSTER** ein:

```
: MUSTER
  ( Plotting Modus - Plotting Modes)
  CLS
  BEGIN
    64 RND 46 RND
    3 PICK PLOT Ø
  UNTIL
;
```

72


AT und PLOT geht nicht
in der untersten Zeile
Spalten
Zeilen
Als Beispiel: das ist
das Pixel (57,32)
Pixel x - Koordinaten
Pixel y - Koordinaten

73

KAPITEL 14

PROGRAMME AUF BAND SPEICHERN

Wenn Sie den ACE abschalten, vergißt er alles, was Sie ihm eingegeben haben. Wollen Sie also das Ergebnis Ihrer Arbeit retten, müssen Sie die Programme auf Kassetten sichern.

Sie benötigen dazu einen normalen Kassettenrecorder, den Sie an den ACE anschließen, wie im Kapitel 3 beschrieben.

Es ist von Vorteil, kurze Kassetten zu benutzen, Sie müssen dann nicht so lange suchen, wenn Sie ein gespeichertes Programm wieder lesen wollen.

Tippen Sie jetzt ein Wort ein, das Sie sichern wollen, z.B.

```
: EINBILDUNG
   CR ." Sie sind bei weitem"
   CR ." das intelligenteste Wesen,"
   CR ." das je einen Jupiter ACE"
   CR ." benutzt hat!"
```

Dann geben Sie ein

```
SAVE HONIG
```

drücken Sie aber noch nicht ENTER.

SAVE erledigt die Speicherung, und HONIG ist der Name, der das Programm auf dem Band identifiziert. Sie müssen nicht extra sagen, daß Sie **EINBILDUNG** speichern wollen, denn **SAVE** speichert alle Worte, die zur Zeit im ACE sind und von Ihnen eingegeben wurden. HONIG ist kein FORTH-Wort, es ist lediglich eine Erkennungsmarke, die sie gewählt haben.

Spulen Sie die Kassette jetzt so weit vor, daß das Magnetband unter dem Schreib-Lesekopf steht. Falls Sie schon Informationen gespeichert haben, müssen Sie natürlich bis an deren Ende vorspulen. Starten Sie jetzt den recorder mit "Aufnahme" und drücken Sie ENTER. Nach etwa 5 Sekunden hören Sie zwei kurze Geräusche aus dem Lautsprecher und auf dem Bildschirm wird OK angezeigt. Damit ist die Aufzeichnung für den Computer abgeschlossen.

Sie sollten sich aber grundsätzlich davon überzeugen, daß die Daten auf der Kassette angekommen sind: Das können Sie mit dem Wort **VERIFY**(=bestätigen, prüfen) tun.

Spulen Sie dazu das Band zurück bis kurz vor die Stelle, wo Sie mit der Sicherung begonnen haben, regeln Sie die Höhen (soweit auf Ihrem Recorder vorhanden) herunter und die Lautstärke auf etwa 3/4.

Dann geben Sie ein:

```
VERIFY HONIG
```

und starten das Band. Sobald das Band Ihre gespeicherten Daten erreicht, kommt eine Nachricht auf dem Bildschirm: "Dict: HONIG" und nach ca. 3 Sekunden OK. In diesem Fall ist alles gut gegangen. Wenn nicht, können Sie den Versuch mit der SPACE-Tasté abbrechen.

Was ist zu tun, wenn die Übertragung mißlingt?

1. Sehen Sie im Kapitel 3 nach und führen Sie alle dort angegebenen Prüfungen durch

74

2. Hören Sie sich die Aufzeichnung über den Lautsprecher des Recorders an. Dazu müssen Sie den Kopfhörer-Stecker herausziehen. Zuerst müßte etwa 5 Sekunden lang ein hohes Pfeifen zu hören sein, danach weniger als eine Sekunde lang ein lauter Ton, dann wieder, aber nur kurz das Pfeifen, und schließlich noch einmal der laute Ton.
   All dies müßte unerfreulich laut sein, wen Sie die Lautstärke des Recorders voll aufdrehen. Hören Sie dagegen nichts, dann ist auch nichts angekommen. Unter Umständen passen die Stecker nicht richtig zum Recorder. Ziehen Sie sie einige Millimeter heraus und wagen Sie einen neuen Versuch.

   Waren die Töne gut zu vernehmen, dann versuchen Sie **VERIFY** bei voller Lautstärke. Bei zu schwacher Lautstärke kann der Computer die Signale nicht einwandfrei empfangen; ist es zu laut, werden sie leicht übersteuert. Machen Sie deshalb mehrere Versuche mit verschiedenen Lautstärke-Einstellungen, bis Sie die richtige für Ihren Recorder gefunden haben.

   Wenn überhaupt nichts geht, haben Sie einen schlechten Tag erwischt (oder ein schlechtes Band). Bauen Sie Ihre Agression an einem Ihrer Familienmitglieder ab und gehen Sie ins Bett. Versuchen Sie es am nächsten Tag mit einem neuen Band, reinigen Sie die Köpfe oder leihen Sie sich ein anderes Gerät.

   Nach den Informationen in Kapitel 3 können Sie Ihr Wörterbuch wieder in den Computer laden. Ein Wörterbuch, das vom Band in den Computer gelesen wird, wird an das Ende eines bereits vorhandenen Wörterbuchs angehängt. Sie können also kleine Gruppen von Wörtern getrennt sichern und nur die zurückholen, die Sie jeweils brauchen. Wollen Sie zuviel auf einmal laden, erhalten Sie ERROR 1Ø. Im Anhang B erfahren Sie mehr über diese Fehlermeldung.

SAVE, VERIFY und **LOAD** werden für Wörterbücher auf Band verwendet. Sie können auch Informationen speichern, indem Sie festlegen, wieviele Bytes Sie sichern wollen, und bei welcher Adresse Sie beginnen. In diesem Fall verwenden Sie **BSAVE, BVERIFY** und **BLOAD.**

Soll zum Beispiel der Bildschirminhalt gespeichert werden, dann geben Sie ein

8192 768 **BSAVE** BILD

Der Bildschirm hat 768 Bytes, beginnend mit Adresse 8192.

BILD ist wiederum nur ein Name, der die Daten auf dem Band identifiziert. **BSAVE** ist genauso zu bedienen wie **SAVE.**

In diesem speziellen Beispiel ist es nicht unbedingt sinnvoll, sofort **BVERIFY** durchzuführen. Durch den Vorgang wird nämlich der Bildschirm selbst verändert. Mit

8192 768 **BLOAD** BILD

dagegen erhalten Sie wieder das ursprüngliche Bild.

Noch einfacher ist es, wenn Sie BILD immer wieder an die gleiche Stelle laden wollen, zu sagen

Ø Ø **BLOAD** BILD

Sie müssen dabei folgende Regeln beachten:

- die erste Zahl (die **BLOAD** an zweiter Stelle im stack findet), ist die Speicheradresse, bei der Sie das Laden anfangen wollen. Es muß ja nicht immer die gleiche sein, ist sie es aber, dann genügt auch Ø als Eingabe.

75

- Die zweite Zahl (die **BLOAD** am TOS findet) ist als reine Vorsichtsmaßnahme gedacht. Sollten Sie nicht mehr wissen, wieviele Bytes gespeichert wurden, kann es gefährlich sein, sie unbesehen wieder zurückzuholen, weil Ihnen plötzlich Speicherplatz überschrieben wird, den Sie noch brauchen. Mit der zweiten Zahl wissen Sie dann aber wenigstens, wieviel Speicherplätze maximal zerstört wurden.
  Geben Sie statt der Zahl eine Null ein, dann wird die ursprüngliche Zahl von Bytes übernommen.

  Beide Regeln gelten für **BVERIFY** und **BLOAD**.

  Wörterbuch-Daten und Bytes-Daten sind durchaus etwas Unterschiedliches. ACE sagt Ihnen, womit er gerade zu tun hat, indem er "Dict:" oder "Bytes:" vor den Datennamen setzt. Mit **LOAD** können Sie keine Bytes-Daten, mit **BLOAD** keine Wörter-Daten laden.

Zusammenfassung:

Band-Daten: Wörterbuch-Daten und Byte-Daten

FORTH-Wörter: **SAVE, VERIFY, LOAD, BSAVE, BVERIFY, BLOAD**

Übungen:

1. Nachdem Sie den Byte-Datensatz BILD (von Adresse 8192 aus) gesichert haben, geben Sie ein:

   9216 Ø BLOAD BILD

   um es an die Adresse 9216 zurückzuladen. Es wird Ihren Bildschirm mehr oder weniger verändern.

   Was heißt das? Wo ist der Bildschirminhalt wirklich gespeichert, in 8192 oder in 9216? Die Antwort heißt: In beiden. Diese Adressen sind gewissermaßen Vorder-und Hintereingang des Bildschirmbereiches

   Der Vordereingang hat die Adressen 8192 bis 9215. Wenn Sie diese ansprechen, werden Sie sofort vom Computer bedient, auch wenn er vorübergehend den Bildschirm außer Acht lassen muß (Sie sehen kurzzeitig weiße Punkte). Da das Band nicht angehalten werden kann, verwendet man diese Adressen beim Sichern oder Laden des Bildes (obwohl es langsam genug läuft, um in der Praxis keine Probleme zu verursachen).

   Der Hintereingang hat die Adressen 9216 bis 1Ø239. Sie werden nicht immer sofort bedient, wenn der Computer gerade mit dem Bildschirm beschäftigt ist. Andererseits erhalten Sie bei Benutzung dieser Adressen keine störenden weißen Punkte.

   Einen ähnlichen Aufbau hat der Speicher für den Zeichensatz. Die Hauptadressen sind 1Ø24Ø bis 11263, die Nebenadressen (die wir im Kapitel 12 benutzt haben) sind 11264 bis 12287.

2. Es ist natürlich zweckmäßig, einen mit viel Mühe selbst erzeugten Zeichensatz für die spätere Verwendung auf Kassette zu speichern. Unglücklicherweise gibt es aber keinen Direktzugriff auf den Zeichensatz (s.Kap.12). Sie können das Problem aber lösen, indem Sie den Zeichensatz an eine andere Stelle im Speicher verlagern (z.B. in den Bildschirmbereich) und von hier aus sichern. Das spätere Zu- rückladen in den Zeichensatz ist möglich.

 76

3. Noch ein hübsches Spielchen. Geben Sie Ihr Lieblingswort ein (EINBILDUNG) und tasten Sie danach in einem Vorgang ein

```
8896 32 BSAVE BILDUNG LOAD HONIG EINBILDUNG
```

Schalten Sie den Recorder ein und drücken Sie ENTER. Dadurch sichern Sie einen Datenbestand BILDUNG. 8896 ist die Adresse des Eingabebereichs, wenn dieser zwei Zeilen lang ist.

Sie haben jetzt BILDUNG gesichert und der Computer sucht nach einer Wörterbuch-Datei HONIG. Unterbechen Sie mit SPACE und sichern Sie das Wörterbuch mit

```
SAVE HONIG
```

Jetzt haben Sie eine Byte-Datei BILDUNG, die eine Zeile des Eingabebereichs umfaßt und aussagt "LOAD HONIG EINBILDUNG", gefolgt von einer Wörterbuch-Datei HONIG. Laden Sie BILDUNG mit

```
8928 Ø BLOAD BILDUNG
```

(Beachten Sie, daß sich die Adresse geändert hat, weil der Eingabebereich nur noch eine Zeile umfaßt).

Der Computer lädt jetzt BILDUNG in den Eingabebereich, lädt danach das Wörterbuch HONIG und führt EINBILDUNG aus.

KAPITEL 15

BRÜCHE UND DEZIMALPUNKTE

Bisher haben wir nur mit ganzen Zahlen (engl.: integers) gearbeitet und in manchen Versionen der FORTH-Programmiersprache dürfen andere Versionen auch gar nicht verwendet werden. Es ist ja auch erstaunlich, wieviel man mit ganzen Zahlen anfangen kann. Aber ACE läßt zusätzlich auch Dezimalbrüche zu, weil es bequemer ist. Solche Zahlen heißen Gleitkommazahlen.

Hier ein Beispiel:

```
2.1 2.1 F+ F.
```

Als "Komma" muß immer der Punkt verwendet werden. Um Sie daran zu erinnern, werden wir im Folgenden vom Dezimalpunkt sprechen (obwohl die Zahlen im deutschen Sprachgebrauch "Gleitkommazahlen heißen)

Die wichtigste Regel ist: Sie müssen dem Computer immer sagen, daß er es mit Dezimalzahlen zu tun hat, und nicht mit ganzen Zahlen. Deshalb weerden anstatt + und . die Wörter **F+** und **F.** verwendet (das "F" kommt vom englischen Floating-point = Gleitkomma).

Versuchen Sie im Vergleich dazu

```
2.1 2.1 + .
```

das "Ergebnis" 16673 ist Unsinn.

Es gibt natürlich auch Dezimal-Versionen für -, *, / und **NEGATE**, alle durch F am Anfang gekennzeichnet: F-, F*, F/ und FNEGATE.

Im übrigen werden sie behandelt wie bereits bekannte FORTH-Wörter.

Alle Dezimalzahlen im ACE FORTH müssen mit Dezimalpunkten geschrieben werden. Das gilt auch für ganze Zahlen, die in Dezimal-Arithmetik verarbeitet werden sollen. Wollen Sie zum Beispiel 11 : 4 mit dem richtigen Ergebnis 2.75 rechnen, müssen Sie dies mit dem Wort F/ tun. Dazu brauchen aber auch beide Zahlen einen Dezimalpunkt

```
11. 4. F/ F.
```

Es gibt eine weitere Schreibweise für Gleitkomma-Zahlen, die sogenannte Exponentialschreibweise. Dabei folgen unmittelbar auf die Zahl (natürlich mit Dezimalpunkt) ein E und eine ganze Zahl. Diese ganze Zahl ist der Exponent zur Basis 10 und bedeutet, daß die Gleitkommazahl so oft mit 10 zu multiplizieren ist, wie der Exponent angibt.

```
2.1E0 = 2.1
2.1E1 = 2.1*10 = 21
2.1E2 = 2.1*10*10 = 210
2.1E3 = 2.1*10*10*10 = 2100
```

Ist der Exponent negativ, wird die Zahl durch 10 dividiert

```
2.1E-1 = 2.1/10 = 0.21
2.1E-2 = 2.1/10/10 = 0.021
2.1E-3 = 2.1/10/10/10 = 0.0021
```

Der Exponent bewirkt eine Verschiebung des Dezimalpunkts um entsprechend viele Stellen.

In Verbindung mit Gleitkommazahlen gibt es noch zwei andere Wörter:

INT wandelt eine Gleitkommazahl in eine ganze Zahl um, indem es die Ziffern nach dem Dezimalpunkt wegläßt

```
12.99 INT .     ergibt 12
-12.99 INT .    ergibt -12
```

UFLOAT wandelt ganze Zahlen in Gleitkommazahlen um:

```
12 UFLOAT F.
```

ergibt 12.

Negative Zahlen werden mit einem besonderen Kniff umgewandelt. Der Computer addiert 65536 dazu und wandelt dann in Gleitkomma um (wozu das gut ist, sehen Sie in Kapitel 17). Es gibt also durch die Umwandlung nie negative Zahlen. Dadurch erklärt sich auch das "U" in **UFLOAT**, es bedeutet im Englischen eine Abkürzung für "unsigned = ohne Vorzeichen).

Bitte beachten Sie:
Die Größe von Gleitkommazahlen im Computer ist begrenzt. Positive Zahlen müssen zwischen 1.ØE-64 und 9.99999E62 liegen. Falls Sie diesen Bereich verlassen, können Rechenergebnisse noch vernünftig aussehen, geben aber weiterhin falsche Resultate.
Negative Zahlen müssen zwischen -9.99999E62 und -Ø.1E-64 liegen.

INT gibt richtige Ergebnisse nur im normalen Bereich der ganzen Zahlen, also zwischen -32768 und 32767.

Bei der Behandlung von Gleitkommazahlen müssen Sie daran denken, daß jede Gleitkommazahl den Platz von zwei ganzen Zahlen braucht. Es ist daher empfehlenswert, sich Wörter für die Gleitkommadarstellung zu definieren:

```
: 2DROP
  ( Gleitkommazahl - )
  DROP DROP
;
: 2DUP
  ( Gleitkommazahl - GK'zahl, GK'zahl)
  OVER OVER
;
: 2SWAP
  ( GK1, GK2, - GK2, GK1)
  4 ROLL 4 ROLL
;
: 2OVER
  ( GK1, GK2 - GK1, GK2, GK1)
  4 PICK 4 PICK
;
: 2ROT
  ( GK1, GK2, GK3 - GK2, GK3, GK1)
  6 ROLL 6 ROLL
;
: 2@
  ( Adresse - GK)
  DUP @ SWAP 2+ @
;
: 2!
  ( GK - Adresse)
  ROT OVER ! 2+ !
;
```

Zusammenfassung:

Gleitkommazahlen

FORTH-Wörter: F+, F-, F*, F/, FNEGATE, INT, UFLOAT

79

Übungen:

1. Definieren Sie Gleitkomma-Versionen **2ROLL** und **2PICK** von **ROLL** und **PICK**.

2. Welche von diesen Beispielen sind richtig?

   2 3 F/ F.

   2 7.6 F+ F.

   2. 2.5 F*.

   2. 3. + .

   (Antwort: keines, korrigieren Sie!)

3. Über die Gleitkomma-Arithmetik ist schon viel geschrieben worden. Nachstehend geben wir zwei Beispiele.

   Quadratwurzel: Das Quadrat einer Zahl ist die Zahl mit sich selbst multipliziert, die Wurzel einer Zahl ist der umgekehrte Vorgang. Sie wollen eine Zahl ermitteln, deren Quadrat eine gegebene Zahl ist. Zum Beispiel ist 16 das Quadrat von 4; 4 ist die Quadratwurzelaus 16. Im allgemeinen kann man Quadratwurzeln nicht genau berechnen, weil es unendliche Dezimalzahlen sind. Die Quadratwurzel von 2 ist ungefähr 1.41421.

   Das Wort **QWURZEL** errechnet die Quadratwurzel einer Gleitkommazahl

```
: QWURZEL
  ( Gleitkommazahl - Wurzel)
  1. 1Ø Ø
  DO
    2OVER 2OVER F/ F+
    .5 F*
  LOOP
  2SWAP 2DROP
;
```

   Hier ist eine Methode verwendet, die schon im Altertum der Grieche Heron benutzt hat.
   Man beginnt mit einer Konstanten und verbessert diesen Wert in jedem DO...LOOP-Durchgang.

   Was geschieht, wenn Sie -2. **QWURZEL** rechnen? -2 kann keine Wurzel haben, weil jedes Quadrat positiv wird. Trotzdem wird -2. **QWURZEL** ein - wenn auch unsinniges - Resultat geben.

4. Das folgende Wort berechnet den Sinus eines Winkels, der im Bogenmaß angegeben wird. Es hat eine Genauigkeit von 3 Stellen für Winkel um 2pi und 5 Stellen für Winkel um pi oder weniger. Es addiert die Elemente der Potenzreihe

$$\sin x = x - \frac{x^3}{3*\ \ 2} + \frac{x^5}{5*4*\ \ 3*2} - \frac{x^7}{7*6*5*\ \ 4*3*2} + \ldots$$

```
: SIN
  ( x - Sinus von x)
  2DUP 2DUP 2DUP F* FNEGATE
  2ROT 2ROT ( -x*x, x, x)
  27 2
  DO
    ( -x*x, bisher aufgelaufene Summe in der Potenzreihe)
    6 PICK 6 PICK ( Kopie von -x*x an den TOS)
    F* I I 1+ *
    UFLOAT F/ ( -x*x, Summe nächstes Glied)
```

Fortsetzung Folgeseite

```
      2DUP  2ROT F+ 2SWAP ( -x*x, nächste Summe, n.Glied)
    2
  +LOOP
  2DROP 2SWAP 2DROP
;
```

Es folgen noch die Wörter für Cosinus und Tangens

```
: COS
  ( x - cos x)
  1.5708Ø 2SWAP F- ( pi/2-x)
  SIN
;

: TAN
  ( x - tan x)
  2DUP SIN
  2SWAP COS F/
;
```

KAPITEL 16

PROGRAMMBEEINFLUSSUNG ÜBER TASTATUR

Bisher haben wir nur mit Zahlen aus dem stack gearbeitet. Die Operanden waren vor dem auszuführenden Wort einzugeben. Es gibt aber auch Methoden, mit denen man während eines Programmablaufs Daten eintasten kann, oder bei denen man Daten verwendet, die erst nach dem Wort eingegeben werden.

Das einfachste Wort ist **INKEY**. Es fragt lediglich die Tastatur ab. Wenn Sie eine Taste betätigen (es werden auch die SHIFTS berücksichtigt), legt **INKEY** deren ASCII-Code auf dem stack ab, andernfalls kommt eine Ø auf den stack.

INKEY kann auf verschiedene Weise eingesetzt werden. In einem schnellen Spielprogramm z.B. werden häufig mit den Tasten 5,6,7 und 8 (Pfeile) Figuren auf dem Bildschirm bewegt. Andere Programme sollen in der Ausführung einfach warten, bis der Benutzer eine Taste betätigt. Ein Wort hierfür ist **WARTE**.

```
: WARTE
  ( - )
  BEGIN
    INKEY
  UNTIL
;
```

Oder Sie wollen eine Taste drücken und deren ASCII-Code im stack behalten

```
: TASTE
  ( - ASCII-Code)
  BEGIN
    INKEY ?DUP
  UNTIL
;
```

Ein komplizierteres Wort fragt "(J/N)" und fordert den Benutzer auf, "J" oder "N" einzugeben.Für J legt es eine 1, für N eine Ø auf dem stack ab.

```
: J/N
  ( - Ø oder 1 )
  ." (J/N)?"
  BEGIN
    INKEY DUP ASCII j =
    IF
      2
    ELSE
      ASCII n =
    THEN
    ?DUP
  UNTIL
  1-
;
```

INKEY liest direkt aus der Tastatur, es kann daher immer nur ein Zeichen erkennen. Andere Wörter arbeiten mit dem ganzen Eingabebereich, so daß auch ganze Wörter eingegeben werden und verarbeitet werden können.

Es folgen zunächst zwei Wörter, die das Programm anhalten. Sie können dann in der gewohnten Weise beliebige Daten eingeben. Nach Drücken der ENTER-Taste fährt das Programm fort.

QUERY löscht zunächst den Eingabebereich, danach können Sie eingeben.

RETYPE läßt den Eingabebereich unverändert, Sie können dessen Inhalt ausgeben. Die MARKE ist anfangs **[**, sie weist damit auf die Änderungsmöglichkeit hin.

Keines dieser beiden Wörter beeinflußt den stack.

Die nächsten Wörter befassen sich mit dem Eingabebereich, wenn Sie etwas eingegeben haben:

Das einfachste Wort ist **LINE**. Es interpretiert den Eingabebereich als Zahlen und FORTH-Wörter wie üblich. Es zeigt ebenfalls ein **?** für ein nicht erkanntes Wort an, arbeitet also genau im üblichen Modus, mit dem Zeilen behandelt werden. Es können also z.B. auch Programme aufgerufen werden. Ist die Eingabe jedoch bearbeitet, fährt **LINE** mit Ihrem Programm fort. Sein Einfluß auf den stack richtet sich völlig nach dem Inhalt des Eingabebereichs.

Das Wort **EINGABE** stellt eine sehr einfache Methode dar, ein Programm anzuhalten, um eine Zahl eingeben zu können. (Wenn Sie BASIC kennen, werden Sie so etwas wahrscheinlich schon vermißt haben).

```
: EINGABE
  ( - ?)
  QUERY LINE
;
```

EINGABE ist äußerst flexibel, denn Sie können eine Zahl, die Sie eingeben wollen, erst durch eine Rechnung im Computer erzeugen lassen. Nehmen Sie an, Sie benötigen das Produkt 32 * 23 für eine Eingabe, haben aber keine Lust, es vorher auszurechnen. Tasten Sie einfach

```
32 23 *
```

und **EINGABE**.

Der Bediener muß aber andererseits sehr gut aufpassen, denn mit diesem Wort kann er leicht mehr als eine Zahl in den stack bringen und dadurch das Programm in der Folge zu falscher Verarbeitung veranlassen.

Wir lassen nun eine modifizierte Version von **EINGABE** folgen, die zwar nicht völlig narrensicher ist, aber doch mehr Sicherheit bietet. Sie bringt die Zahl -32768 auf den stack und prüft hinterher, ob sie noch vorhanden ist.

```
: EINGABE
  ( - ?)
  -32768 QUERY INVIS LINE VIS SWAP -32768 -
  IF
    ." JETZT HABEN SIE ABER"
    CR ." NICHT AUFGEPASST"
    QUIT
  THEN
;
```

Wir haben die Wörter **INVIS** und **VIS** hier eingesetzt, weil **LINE** üblicherweise den Eingabebereich in den oberen Bereich des Bildschirms kopiert (auch wenn es nicht OK sagt). In einem Programm könnte das unter Umständen störend sein.

Beachten Sie das Wort **QUIT**, das im Fehlerfall angewandt wird. **QUIT** verläßt das gesamte Programm, also nicht nur **EINGABE**, sondern alle zur Zeit gerade ausgeführten Wörter. Es versetzt den Computer in den normalen Eingabezustand zurück. Beachten Sie auch, daß **QUERY LINE** in **EINGABE** den Eingabestatus nur imitiert, **QUIT** aber in den echten Eingabezustand zurückführt.

83

QUIT löscht den stack nicht. Dafür gibt es ein Wort **ABORT**, das wie **QUIT** wirkt, jedoch auch den Inhalt des stack beseitigt.

Sie können den Eingabebereich besser unter Kontrolle halten, wenn Sie ihn Wort für Wort analysieren. Dafür stehen Ihnen **NUMBER**, **FIND** und **WORD** zur Verfügung. Jedes sucht nach einem bestimmten Begriff, nimmt ihn vom Anfang des Bereichs weg und kopiert ihn (trotz des eingeschalteten **INVIS**) in den oberen Bildschirmteil.

NUMBER sucht eine Zahl (Ganz- oder Gleitkommazahl) am Bereichsanfang. Steht dort keine Zahl, legt es einfach eine Ø auf den stack. Wird eine Zahl gefunden, erfolgt die Anzeige auf dem Bildschirm, die Zahl wird auf dem stack abgelegt, und darüber (am TOS) noch durch ein Kennzeichen ergänzt, welches aussagt, ob es sich um eine ganze Zahl (Kennzeichen 41Ø2) oder eine Gleitkommazahl (Kennzeichen 4181) handelt.

Mit dem folgenden Wort **INTEGER** kann man eine ganze Zahl aus dem Eingabebereich holen. Ist keine vorhanden, wird mit **RETYPE** erneut eine Eingabemöglichkeit geboten:

```
: INTEGER
  ( - Ganzzahl)
  BEGIN
    RETYPE NUMBER DUP 4181 =
    IF
    ( Gleitkommazahl)
    ." UNGUELTIG"
    DROP DROP DROP Ø
   THEN
  UNTIL
;
```

Das Wort **FIND** kann Wörter identifizieren, die im Wörterbuch gespeichert sind. Jede Wortdefinition braucht einen gewissen Platz im Speicher und hat eine Adresse (der technische Begriff dafür ist Umwandlungs-Adresse, engl. Compilation Adress). Ein definiertes Wort am Anfang des Eingabebereichs wird nach oben kopiert und seine Umwandlungsadresse im stack abgelegt. Mit

```
FIND DUP .
```

erhalten Sie die Anzeige der Umwandlungsadresse von **DUP**.

Die Wortfolge **FIND DUP .** arbeitet folgendermaßen:

1. ACE sucht nach einem auszuführenden Wort und findet zunächst **FIND** Er kopiert es nach oben. Zu diesem Zeitpunkt steht im Eingabebereich noch

   DUP .

2. Jetzt wird **FIND** ausgeführt. Es sucht wiederum nach einem Wort, dessen Umwandlungsadresse es auf den stack legen kann. Es findet **DUP**, kopiert es nach oben, und im Eingabebereich bleibt nur noch .

   zurück.

3. Nach **FIND** führt ACE das nächste Wort . aus. Er kopiert . (der Eingabebereich ist jetzt leer), führt es aus, indem er die Adresse von **DUP** anzeigt.

4. Danach gibt es nichts mehr zu tun. ACE zeigt OK an, und wartet auf neue Eingaben.

84

Ein Wort, das die Umwandlungsadresse verwendet, ist **EXECUTE** (Adresse -). Es nimmt die Adresse vom stack und führt das zugehörige Wort aus. Denken Sie daran, daß sich durch **REDEFINE** Umwandlungsadressen ändern können.

Mit dem Wort **WORD** können Sie auch "sinnlose" Wörter behandeln, die weder Zahlen noch Definitionen sind. Da Leerstellen (SPACES) nie Teil eines Wortes sind, sondern im Gegenteil zur Trennung von Wörtern benutzt werden, kann **WORD** das erste Wort im Eingabebereich feststellen. Außer auf den Bildschirm wird das Wort noch in einen Arbeitsbereich im Speicher kopiert. Dieser Bereich heißt "Auffüllbereich" (Pad). Er wird zunächst mit Leerstellen (SPACES) aufgefüllt. Er belegt 254 Bytes im RAM und beginnt bei Adresse 9985. Seine Aufgabe ist es, die Verarbeitung von Texten zu ermöglichen. Mit dem Wort **PAD** kann die Anfangsadresse 9985 auf den stack gelegt werden.

Was tut **WORD** ?
Erstens wird der Auffüllbereich mit Leerstellen gefüllt.

Zweitens nimmt es einen Operanden vom stack, der die Wortbegrenzung (=delimiter) darstellt. Jedes Zeichen kann als Wortbegrenzer verwendet werden. Dazu ist nur sein ASCII-Code vor Ausführung von **WORD** auf dem stack abzulegen. Für Leerstellen ist der Code 32, andere Codes definieren Sie bitte mit ASCII.

Drittens sucht **WORD** einen Text am Anfang des Eingabebereichs. Begrenzer vor dem Text werden übergangen. Der Text wird bis zum nächsten Begrenzer oder zum Bereichsende gelesen.

Viertens überträgt es den Text einschließlich Begrenzer in den Auffüllbereich (Pad) ab Adresse 9986. Die Wortlänge (ohne Begrenzer) wird ins erste Byte des Pad eingetragen (Adresse 9985).

Fünftens legt es die Adresse 9985 auf den stack.

Geben Sie z.B. ein

```
32 WORD axolotl .
```

Ein solches Wort wurde nie definiert. Das ist aber gleichgültig. Der . druckt 9985 aus, die Zahl vom stack.
Im Auffüllbereich sind die ersten Bytes

Adresse		
9985	7	Länge von "axolotl"
9986	97	ASCII-Code für a
9987	120	ASCII-Code für x
9988	111	ASCII-Code für o
9989	108	ASCII-Code für l
9990	111	ASCII-Code für o
9991	116	ASCII-Code für t
9992	108	ASCII-Code für l
9993	32	ASCII-Code für space
	.	
	.	
	.	

Mit **C@** . können Sie selbst nachsehen, mit

```
9986 7 TYPE
```

das ganze Wort "axolotl" anzeigen.

 85

Von einem gespeicherten Text müssen wir Adresse und Länge kennen. Diese Information kann man mit folgenden Schritten erhalten:

1. Wenn **WORD** den Text in den Auffüllbereich überträgt, stellt es dessen Länge voran und legt die Adresse im stack ab.
2. In vielen FORTH-Versionen gibt es ein Wort **COUNT**, mit dem man aus den Informationen aus Punkt 1 die für TYPE (Pkt.3) notwendigen Daten ermitteln kann (Adresse - Adresse + 1, Länge).
   Wir definieren es selbst

```
: COUNT
  DUP 1+ SWAP C@
;
```

3. TYPE nimmt Adresse und Länge vom stack und zeigt den stack an.

Es folgt ein Beispiel, das **WORD** und **COUNT** benutzt. Es übernimmt eine eingegebene Nachricht und schiebt sie über den Bildschirm.

```
: NACHRICHT
  ( - )
  ASCII ∿ WORD CLS
  BEGIN
    32 Ø
    DO
      1Ø I AT SPACE DUP
      COUNT TYPE ( schiebt die Nachricht um eine Stelle
                   nach rechts)
      11 Ø AT 32 SPACES ( löscht die Teile, die in die
                          Folgezeile geschoben werden)
      15ØØ Ø
      DO
      LOOP
    LOOP
    Ø
  UNTIL
;
```

Wir haben ∿ als Begrenzer gewählt, damit die Nachricht Leerstellen enthalten kann, z.B.

NACHRICHT Hallo Nachbarn! ∿

Wenn das Zeichen ∿ fehlt, werden die Leerstellen am Ende des Eingabebereichs als Teil der Nachricht behandelt.

Noch einmal zur Erinnerung: Ob der Eingabebereich Informationen aus Ihrer letzten Eingabe enthält (wie bei Nachricht), oder ob das Programm anhält, damit Sie Daten eingeben können (wie in **INTEGER**): LINE, NUMBER, FIND und **WORD** arbeiten immer in der gleichen Weise.

Zusammenfassung:

Auffüllspeicher (Pad) und Eingabebereich.

FORTH-Wörter **INKEY**, **QUERY**, **RETYPE**, **LINE**, **QUIT**, **ABORT**, **NUMBER**, FIND, **EXECUTE**, **WORD**, PAD

Übungen:

1. Viele FORTH-Dialekte enthalten ein Wort **-TRAILING** (Adresse, Länge mit Leerstellen - Adresse, Länge ohne Leerstellen). **-TRAILING** beginnt mit Adresse und Länge eines beliebigen Textes (wie er von **COUNT** erzeugt wird) und verändert seine Länge, indem es alle Leerstellen am Anfang des Textes entfernt.
   Schreiben Sie eine Definition für **-TRAILING**. Stellen Sie aber sicher, daß es auch dann richtig arbeitet, wenn der Text nur aus Leerstellen besteht.

86

2. Definieren Sie ein Wort

```
: PCT
  PAD COUNT TYPE
;
```

Sie können **PCT** einsetzen, um festzustellen, was am Anfang des PAD steht. Untersuchen Sie, was die folgenden Wörter mit dem Pad anstellen: **ASCII**, **SAVE**, **LOAD**, **:** . Sie alle wirken auf den Pad, so daß ein Text dort nicht immer sicher ist. Auch **.** benutzt den Pad, aber am anderen Ende.

3. Wenn der Computer gerade einmal leer ist, dann definieren Sie

```
: EWIG
  BEGIN
    QUERY Ø
  UNTIL
;
```

Das Wort ist nur sehr schwer zu unterbrechen, denn mit ENTER kommen Sie aus QUERY heraus, müssen aber BREAK drücken, bevor **QUERY** wieder angesprochen wird. Versuchen Sie es, indem Sie SHIFT festhalten und ENTER und SPACE fast gleichzeitig drücken. Noch besser ist es allerdings, ein solches Wort garnicht erst zu definieren.

87

KAPITEL 17

ZAHLENSYSTEME

Wir sind es gewohnt, im Zehnersystem zu rechnen und zu zählen. Dazu stehen uns zehn Ziffern (Ø bis 9) zur Verfügung. Es ist anzunehmen, daß Menschen deshalb im Zehner- oder Dezimalsystem arbeiten, weil sie zehn Finger haben.

Der Computer hat keine Finger. Er kann aber zwei Zustände unterscheiden: Ein Schalter ist entweder ein- oder ausgeschaltet, eine elektrische Spannung ist vorhanden oder nicht vorhanden. Diese beiden Zustände können wir mit Ziffern definieren: "Schalter ein" bedeutet 1, "Schalter aus" bedeutet Ø. Vielleicht leben auf einem fernen Planeten Lebewesen, die an jeder Hand 8 Finger haben. Die würden dann folgerichtig mit einem Sechzehnersystem arbeiten.

In der Tat kann man mit jedem beliebigen Zahlensystem rechnen. Man braucht dazu nur soviel verschiedene Ziffernzeichen, wie das System angibt

- im Zweiersystem sind das die Ziffern Ø und 1,
- im Zehnersystem die Ziffern Ø bis 9, und
- im Sechzehnersystem Ø bis 9 und noch sechs weitere Zeichen, die wir der Einfachheit halber mit den Buchstaben A bis F definieren wollen.

Hat man nun bis zur letzten Ziffer gezählt, dann stellt man einfach eine Stelle voran und zählt weiter.

Die folgende Tabelle soll veranschaulichen, wie die Zahlen in den verschiedenen Systemen dargestellt werden können.

Zahl	Dezimalsystem	Dualsystem	Hexadezimalsystem
Null	Ø	Ø	Ø
Eins	1	1	1
Zwei	2	1Ø	2
Drei	3	11	3
Vier	4	1ØØ	4
Fünf	5	1Ø1	5
Sechs	6	11Ø	6
Sieben	7	111	7
Acht	8	1ØØØ	8
Neun	9	1ØØ1	9
Zehn	1Ø	1Ø1Ø	A
Elf	11	1Ø11	B
Zwölf	12	11ØØ	C
Dreizehn	13	11Ø1	D
Vierzehn	14	111Ø	E
Fünfzehn	15	1111	F
Sechzehn	16	1ØØØØ	1Ø

88

Mit dem Computer können Sie jedes beliebige Zahlensystem darstellen. Geben Sie z.B. ein:

```
2 BASE C!
```

und schon können Sie 1Ø und 1Ø zu 1ØØ addieren

```
1Ø 1Ø + .
```

BASE ist eine Variable, die der Computer bereithält, also eine Systemvariable. Anders als eine normale Variable hat sie jedoch nur 1 Byte und anstelle von **@** und **!** müssen Sie **C@** und **C!** benutzen, um sie ansprechen zu können. Ihr Wert ist die Zahlenbasis (das Zahlensystem), die Sie gerade benutzen. Haben Sie also mit **BASE** die Basis auf 2 gesetzt, müssen Sie Ihre Zahlen auch in der Binär-Schreibweise eingeben. Genauso antwortet dann auch der Computer. (Bei Gleitkomma-Zahlen ist es etwas anders, sie haben immer die Basis 1Ø).

Um zur menschlichen Schreibweise zurückzufinden, müssen Sie

```
1Ø1Ø BASE C!
```

eingeben, denn 1Ø1Ø ist im Zweiersystem, das ja noch aktiv ist, die 1Ø des Dezimalsystems. Das gleiche können Sie mit dem Wort DECIMAL erreichen, es setzt die Basis auf 1Ø.

Unser gewohntes Zahlensystem ist das Dezimalsystem, das System mit der Basis zwei heißt Dual- oder Binärsystem, das System auf der Basis 16 heißt Hexadezimalsystem oder einfach Hex.

Der Jupiter ACE ist für verschiedene Zahlensysteme entwickelt, je nachdem, wie die Basis definiert wird. Ein dreizehiges Faultier würde vielleicht die Basis 3, und ein einarmiger Seeräuber die Basis 5 wählen.

Wie stellen wir nun fest, welchen Wert eine Binärzahl darstellt? In der Dezimalschreibweise bedeuten die verschiedenen Stellen Zehnerpotenzen

```
---------------Hunderter
| ------------Zehner
| | |---------Einer
2 5 5
```

255 bedeutet also: 2 Hunderter + 5 Zehner + 5 Einer.

Das gleiche Prinzip wird auch im Binärsystem angewendet. Die Spalten sind natürlich Zweierpotenzen

```
--------------------Einhundertachtundzwanzig
| -------------------Vierundsechzig
| | ----------------Zweiunddreißig
| | | --------------Sechzehn
| | | | ------------Acht
| | | | | ---------Vier
| | | | | | -------Zwei
| | | | | | | -----Eins
1 1 1 1 1 1 1 1
```

Im Binärsystem bedeutet 11111111:

Dezimal	Binär
128	ein * einhundertachtundzwanzig
64	+ ein * vierundsechzig
32	+ ein * zweiunddreißig
16	+ ein * sechzehn
8	+ ein * acht
4	+ ein * vier
2	+ ein * zwei
1	+ ein * eins
255	= zweihundertfünfundfünfzig

Schneller können Sie rechnen, wenn Sie in Gedanken 1 addieren, damit erhalten Sie die Binärzahl 1 0000 0000. Die 1 gehört in die Spalte "Zweihundertsechsundfünfziger", 1111 1111 ist also 1 weniger = 255.

Der Computer arbeitet, wie wir jetzt wissen, mit dem Binärsystem. Deshalb sind uns bisher auch so häufig Zweierpotenzen begegnet.
Ein Byte z.B. ist eine Zahl, die mit 8 Bits dargestellt werden kann:

Die kleinste Zahl ist Null = 0000 0000

die größte Zahl ist 255 = 1111 1111

Eine ganze Zahl wird im ACE durch zwei Bytes dargestellt. Die größte in diesem Fall darstellbare Zahl besteht also aus 16 Bits und heißt 1111 1111 1111 1111 (oder 65535 in unserer Schreibweise). Das ist auch die größtmögliche Speicheradresse im System.

Nun müssen wir aber positive und negative Zahlen darstellen können. Im ACE wird das so gelöst:

- Der Zahlenbereich von 0 bis 32767 wird in unveränderter Form gespeichert,
- der Zahlenbereich von 32768 bis 65535 repräsentiert die negativen Zahlen von -32768 bis -1.

Wenn Sie zu einer negativen Zahl 65536 addieren, erhalten Sie die im ACE gespeicherte entsprechende Zahl.
Beispiel:

Sie geben ein	-19
ACE addiert	65536
und speichert	65517

Diese Speichermethode heißt Zweier-Komplement-Methode. Sie bedeutet eigentlich, daß wir zwei verschiedene Zahlenbereiche behandeln können

- entweder Zahlen mit Vorzeichen von -32768 bis 32767,
- oder Zahlen ohne Vorzeichen von 0 bis 65535.

Wie wir die Zahlen definieren, hängt von den Umständen ab.

Es gibt ein Wort **U.** mit dem man eine Zahl als Zahl ohne Vorzeichen ausdrucken kann. Unsere Zahl -19 wurde von . als -19 wieder ausgegeben, auch wenn im Speicher 65517 steht. Mit U. dagegen wird sie in ihrer gespeicherten Form als 65517 angezeigt.

90

Ein anderes Wort, das mit Zahlen ohne Vorzeichen arbeitet, ist **U<** (Zahl,Zahl - MERKER). Es entspricht dem Wort < . Geben Sie ein

```
1 -1 < .
1 -1 U< .
```

U< behandelt die -1 als 65535, das ja größer als 1 ist.

Zum Schluß noch ein einfaches Verfahren, mit dem man Binär- in Hex-Zahlen umwandeln kann: Teilen Sie eine gegebene Binärzahl, von rechts beginnend, in Vierergruppen ein und ersetzen Sie dann die Gruppen nach der folgenden Tabelle durch die zugeordnete Ziffer oder den entsprechenden Buchstaben.

Bit-Gruppen	ersetzt durch
ØØØØ	Ø
ØØØ1	1
ØØ1Ø	2
ØØ11	3
Ø1ØØ	4
Ø1Ø1	5
Ø11Ø	6
Ø111	7
1ØØØ	8
1ØØ1	9
1Ø1Ø	A
1Ø11	B
11ØØ	C
11Ø1	D
111Ø	E
1111	F

Für die Zahl hundert z.B. wird

aus Binär Ø11Ø Ø1ØØ

Hex 6 4 , also Hex 64.

Zusammenfassung:

Zahlensysteme: Dezimal, Hexadezimal, Binär

Zahlen mit und ohne Vorzeichen

FORTH-Wörter: **BASE, DECIMAL, U. U<**

Übungen:

1. Warum ist das Binärsystem für die Definition neuer graphischer Zeichen sehr nützlich?

2. Versuchen Sie

```
BASE C@ .
```

Ändern Sie die Zahlenbasis und versuchen Sie es nochmal. Warum bleibt die Basis immer 1Ø, gleichgültig wie oft Sie sie ändern?

Schreiben Sie ein Wort . BASIS, das die Zahlenbasis in Dezimalform druckt. Stellen Sie sicher, daß die Zahlenbasis nach . BASIS wieder gleich ist wie vorher.

91

3. Wählen Sie die Zahlenbasis 36. Sie können jetzt alle Ziffern und Buchstaben als Ziffern benutzen. Jedes undefinierte Wort aus 3 Zeichen wird jetzt als Zahl gespeichert, und Sie können es wie Zahlen verarbeiten, z.B.

```
EIN AUS . .
```

4. Eine Zahl wird negativ dargestellt, indem man sie von 65536 subtrahiert. Im Binärsystem geht das sehr einfach: Subtrahieren Sie von 65535 (das sind lauter Einsen) und addieren Sie 1. Es kommen nur zwei Fälle vor: 1 - 1 = Ø und 1 - Ø = 1.

5. Wenn Sie Hex öfter benutzen wollen, definieren Sie

```
: HEX
    16 BASE C!
;
```

6. Errechnen Sie $2^{1Ø}$ (1Ø24). Da diese Zahl dicht bei 1ØØØ liegt, nennt man sie auch Kilo (imGegensatz zum "k" in km oder kg steht hier "K"). Man spricht in der Datenverarbeitung von einem Kilo-Byte, das sind dann eben nicht 1ØØØ, sondern 1Ø24 Bytes.

KAPITEL 18

BOOLESCHE ALGEBRA

Wir haben in Kapitel 9 gesehen, wie **IF** Zahlen als Bedingungen interpretiert, die entweder "wahr" (nicht Null) oder "falsch" (Null) sind.

Mit den Wörtern **AND**, **OR** und **XOR** können wir Bedingungen kombinieren, wir müssen nur sicherstellen, daß es sich um wirkliche MERKER (nämlich Ø oder 1) handelt. Diese Wörter heißen auch Boolesche Operationen nach dem Mathematiker George Boole.

Nehmen Sie an, Sie hätten in einem Wort zwei Bedingungen und wollen nur dann etwas tun, wenn beide wahr sind, d.h. wenn die erste Bedingung wahr ist und die zweite Bedingung wahr ist. Dafür benutzen Sie das Wort **AND** (=und). Eine Zahl im TOS liegt zwischen 1Ø und 2Ø, wenn sie größer als 9 und kleiner als 21 ist. Zunächst bringen wir die Aussage aus beiden Bedingungen in den TOS durch

```
DUP 9 > SWAP 21 <
```

Jetzt sind im stack zwei MERKER. **AND** ersetzt sie durch 1, wenn sie beide 1 sind. Andernfalls stellt **AND** eine Ø in den TOS. Nach **AND** können Sie **IF** verwenden und wie gewohnt weiterarbeiten.

Wir können diesen Sachverhalt in einer Tabelle darstellen:

TOS	2.v.oben		Ergebnis
Ø	Ø	→	Ø
Ø	1	→	Ø
1	Ø	→	Ø
1	1	→	1

OR (=oder) wird verwendet, wenn entweder die erste oder die zweite Bedingung wahr ist, oder wenn beide wahr sind. Die Umgangssprache verwendet das Wort "oder" in etwas anderer Bedeutung, nämlich im Sinne etwa von Geldstrafe oder Haft. Bei einer solchen umgangssprachlichen "oder"-Verbindung würden Sie sicherlich erwarten dürfen, daß Ihnen nur eines von beiden droht, nämlich Geldstrafe oder Haft.

Die Tabelle für **OR** hat folgende Form:

TOS	2.v.oben		Ergebnis
Ø	Ø	→	Ø
Ø	1	→	1
1	Ø	→	1
1	1	→	1

Das dritte Wort ist **XOR** (es bedeutet "Exclusiv oder"). Es wirkt wie **OR**, das Ergebnis ist jedoch "falsch", wenn beide Bedingungen wahr sind. Das heißt in anderen Worten: Ein "wahres" Ergebnis erhält man nur dann, wenn nur ein Operand "wahr" ist, nicht aber beide. **XOR** entspricht also unserer Bedeutung in der Umgangssprache wesentlich besser.

Die Tabelle für **XOR** hat folgende Form:

TOS	2.v.oben		Ergebnis
Ø	Ø	→	Ø
Ø	1	→	1
1	Ø	→	1
1	1	→	Ø

93

AND, OR und **XOR** arbeiten nur richtig, wenn sie im stack Ø bezw. 1 vorfinden. Was geschieht, wenn dort andere Zahlen stehen?

Nehmen wir an, im stack stünden die Zahlen 1Ø8 und 18, die beide ungleich Ø, also im bisherigen Sinne "wahr" sind. Das Ergebnis von **AND** müßte dann auch "wahr" sein.

AND bearbeitet die Zahlen Bit für Bit:

```
        ØØØØ ØØØØ Ø11Ø 11ØØ      (1Ø8)
AND     ØØØØ ØØØØ ØØØ1 ØØ1Ø      ( 18)
        ØØØØ ØØØØ ØØØØ ØØØØ
```

Dieses Ergebnis bedeutet aber "falsch"!

Die bitweise Verarbeitung von Zahlen mit **AND,OR** oder **XOR** können wir dazu benutzen, Zahlen auf eine besondere Weise miteinander zu kombinieren.
Werden z.B. die Zahlen 1Ø6 (Binär 11Ø 1Ø1Ø) und 2Ø1 (Binär 11ØØ 1ØØ1) durch **AND** miteinander verknüpft, entsteht daraus

```
        ØØØØ ØØØØ Ø11Ø 1Ø1Ø
AND     ØØØØ ØØØØ 11ØØ 1ØØ1
        ØØØØ ØØØØ Ø1ØØ 1ØØØ
```

das ist die Zahl 72.
Genauso sieht es bei **OR** und **XOR** aus.

Hier ist ein typisches Beispiel, in dem **AND** auf recht unterschiedliche Weise verwendet wird. Wir definieren ein Wort **BUCHST**, das einen ASCII Code daraufhin untersucht, ob er einen Buchstaben darstellt, und für "wahr" eine 1, für "falsch" eine Ø im TOS ablegt.

Die ASCII Codes der Großbuchstaben liegen zwischen 65 und 9Ø, die der kleinen Buchstaben zwischen 97 und 122.

```
: BUCHST
  ( ASCII Code - MERKER)
  DUP ASCII A 1- >
  OVER ASCII Z 1+ < AND
  SWAP DUP ASCII a 1- >
  SWAP ASCII z 1+ < AND OR
;
```

Es gibt aber einen hübschen Trick, der zeigt, wie man mit ASCII Codes spielen kann. Der ASCII Code eines Kleinbuchstabens ist um 32 größer als der des entsprechenden Großbuchstabens. In der Binärschreibweise erkennen Sie, daß es genügt, ein Bit von 1 auf Ø zu ändern, um aus einem kleinen einen großen Buchstaben zu machen:

der Binärcode von A ist Ø1ØØ ØØØ1

der Binärcode von a ist Ø11Ø ØØØ1
↑
Dieses Bit ist zu ändern

Mit **AND** und der Zahl 11Ø1 1111 (Dezimal 223) können wir das entscheidende Bit ändern: 7 Bits bleiben so, wie sie waren, das Bit, worauf es ankommt, wird zu Ø.

Wir erhalten auf diese Weise ein kürzeres Wort **BUCHST**. Zunächst ändern wir in Großbuchstaben und fragen nur noch danach:

```
: BUCHST
  ( ASCII Code - MERKER)
  223 AND
  DUP ASCII A 1- >
  SWAP ASCII Z 1+ < AND
;
```

94

Zusammenfassung:

FORTH-Wörter: **AND**, **OR**, **XOR**

Übungen:

1. Obwohl wir x-mal darauf hingewiesen haben, daß **AND**, **OR** und **XOR** nur mit den Operanden Ø und 1 einwandfrei arbeiten, gilt dies für **OR** nicht. Hier kann in der Tat eine "wahre" Bedingung aussehen, wie sie will. Warum?

2. Definieren Sie ein Wort **&**, das zwei Zahlen aus dem stack nimmt und eine 1 hinterläßt, wenn sie "wahr", d.h. ungleich Ø sind, und eine Ø, wenn eine von ihnen Ø ist. (Verwenden Sie **IF..ELSE.. THEN**).
   Definieren Sie auch Wörter, die so mit **OR** und **XOR** korrespondieren, wie **&** mit **AND**.

3. Sie können sich die Zahl 223 in **BUCHST** als eine Art Maske vorstellen, die **AND** vorschreibt, bestimmte Bits eines anderen Wortes unverändert zu lassen und nur eines zu Ø zu machen.

   Entsprechende Masken könnten **OR** dazu veranlassen, bestimmte Bits unverändert zu lassen und den Rest in 1 zu ändern; oder mit **XOR** bestimmte Bits umzukehren, unabhängig davon, wie sie vorher aussahen.

   Stellen Sie hierzu Überlegungen an.

   -1 **XOR** nimmt den TOS und kehrt alle seine Bits um. Aus Ø wird 1, aus 1 wird Ø. Wie geht das? (Hinweis: Wie sieht -1 in Binärform aus?)

KAPITEL 19

ERWEITERTE ARITHMETIK

In einem Byte lassen sich 256 (= 2^8) Zahlen darstellen, in zwei Bytes 256 * 256 = 65536 Zahlen.(Ohne Vorzeichen der Bereich von Ø bis 65535,mit Vorzeichen von -32768 bis 32767).

In vier Bytes könnte man 256^4 = 4 294 967 295 Zahlen unterbringen, dazu müßte man ein Paar normaler Zwei-Byte-Zahlen benutzen. Man erhält damit eine wesentlich höhere Genauigkeit bei vielen Berechnungen. Das ist das Prinzip der Doppellängen-Arithmetik.

Die hierfür vorgesehenen Wörter behandeln die beiden obersten Zahlen im stack wie die beiden Hälften einer 4 Bytes langen Zahl.

D+ (d1,d2 - d1 + d2) addiert zwei doppelt lange Zahlen in den TOS.

Für den Anfang nehmen wir etwas Einfaches, was man mit einfacher Länge wesentlich eleganter rechnen könnte: 3 + 5.

```
3 Ø 5 Ø D+ . .
```

Zunächst werden also für die Eingabe einer doppelt langen Zahl zwei einfache Zahlen eingetastet: 3 Ø für 3 und 5 Ø für 5. In hexadezimalen Ziffern hat die 3 folgendes Aussehen:

```
        ØØØØ  | ØØØ3

signifikantere, | weniger signifikante,
obere Hälfte,   | untere Hälfte,
höher im stack  | tiefer im stack
```

Für diese Vorgänge ist die Zahlenbasis Hex natürlich sehr nützlich, weil die beiden einfachlangen Zahlen, die Sie auf dem stack ablegen wollen, den ersten vier (im stack obenliegenden) und den zweiten vier (im stack danach folgenden) hexadezimalen Ziffern entsprechen. Eine Zahl in Hex (z.B. 4C83A2) muß zunächst auf 8 Stellen erweitert werden:

```
ØØ4C 83A2
```

Nach BASE ! geben Sie ein

```
83A2 ØØ4C
```

Die Ziffern sind gleich, die beiden Gruppen werden bei der Eingabe jedoch vertauscht, damit sie in der richtigen Reihenfolge auf dem stack ankommen.

Beachten Sie bitte: bevor Sie von einer negativen Zahl das Zweierkomplement bilden, müssen Sie noch einige Nullen vor der Zahl einfügen. Das Zweierkomplement wird genauso gebildet wie bei einfachlangen Zahlen (s.Kap.17): 1 wird gegen Ø, und Ø gegen 1 vertauscht (oder die Zahl von Hex FFFF FFFF subtrahiert), und dann 1 addiert.

Haben Ihre Zahlen einfache Länge, dann brauchen Sie die Hex-Schreibweise nicht, auch wenn sie doppeltlang verarbeitet werden sollen. Um die doppelte Länge zu erzeugen, müssen Sie folgendes tun:

1. Bringen Sie die Zahl in den stack.
2. Ergänzen Sie Ø (für positive Zahlen oder Ø), oder -1 (für negative Zahlen)

Einige weitere Wörter für doppelt lange Arithmetik:

D< (d1,d2 - MERKER). Ergibt 1 (wahr), wenn d1 < d2.

DNEGATE (d - -d) kehrt das Vorzeichen der Zahl um.

U* (n1,n2 - n1*n2) führt eine Multiplikation aus. n1 und n2 sind einfachlange Zahlen ohne Vorzeichen. Das Produkt n1*n2 hat ebenfalls kein Vorzeichen, ist aber doppelt lang.

256 256 **U*** . .

ergibt 1 und Ø, die beiden Teile der Zahl 65536 (Hex 1 ØØØØ)

U/MOD (d1,n2 - Rest d1/n2,Quotient). Durch das U wird erreicht, daß alle Zahlen ohne Vorzeichen verarbeitet werden. Auch hier ist, wie in **U***, doppelt lange Arithmetik enthalten. d1, der Dividend,wird doppelt lang, alle anderen Zahlen haben einfache Länge.

/* (n1,n2,n3 - (n1*n2)(n3)). Wir haben dieses Wort schon im Kap.5 behandelt. n1*n2 wird als doppelt lange Zahl ausgerechnet. Dadurch wird die notwendige Genauigkeit erzielt, auch wenn Sie nie eine doppelt lange Zahl zu Gesicht bekommen. */MOD arbeitet entsprechend.

Wir zeigen Ihnen jetzt ein paar Möglichkeiten, Zahlen formatiert darzustellen. Zur Formatierung braucht FORTH doppelt lange Zahlen. Einfach lange Zahlen müssen durch Vorausstellen von Ø bezw. -1 erst auf doppelte Länge umgewandelt werden.

Zur formatgerechten Ausgabe zerlegt FORTH die Zahlen in einzelne Stellen und speichert sie rückwärts in einem Zwischenspeicher, dem Pad. Bei diesem Vorgang können Texte und Sonderzeichen eingefügt werden.

Der Umkehrvorgang kann relativ einfach bewerkstelligt werden. Um etwa die Zahl 123 umzukehren, wird sie durch 1Ø (die Zahlenbasis) dividiert. Man erhält Quotient 12 und Rest 3. Dieser Rest ist die letzte Ziffer. Durch Wiederholung des Vorgangs mit der Zahl 12 erhalten wir Quotient 1 und Rest 2, die nächste Ziffer. Schließlich bleibt noch die 1, die genauso als Quotient ermittelt werden kann. Diese Methode läßt sich in jeder Zahlenbasis anwenden.

Mit <# beginnt der Vorgang.

Mit #> ist er beendet. Die Zahl ist aus dem stack verschwunden. Stattdessen sind dort die Adresse der ersten Ziffer und die Zahl der Ziffern eingetragen (genau was TYPE braucht).

Zwischen diesen beiden Wörtern gibt es:

(d1 - d2) erzeugt eine Ziffer aus einer doppelt langen Zahl (ohne Vorzeichen). Die Zahl wird durch die aktuelle Zahlenbasis dividiert, der Rest in den Zwischenspeicher (er ist die nächste anzuzeigende Zahl), der Quotient bleibt im stack.

#S (d1 - Ø,Ø) wiederholt # solange, bis aus der doppelt langen Zahl im stack Ø geworden ist. Beachten Sie bitte, daß #S eine Ø auf dem stack hinterläßt.

Mit **HOLD** können Sie die gewünschten Sonderzeichen einfügen. **HOLD** (ASCII-Code -) speichert ein Zeichen, dessen ASCII-Code gegeben ist, in den Zwischenspeicher.

Mit dem folgenden Wort **GELD** können Sie einen Betrag, der im stack als Wert in Pfennigen gespeichert ist, formatiert anzeigen:

```
: GELD
  ( Pfennige - )
  Ø ( erzeugt doppelte Länge)
  <# # # ( zwei Stellen für Pfennige)
  ASCII , HOLD ( Komma)
  #S ( Mark)
  ASCII M HOLD ASCII D HOLD
  #> TYPE
;
```

Da alles von rückwärts aufgebaut wird, kommen Pfennige vor DM, und auch im Wort DM zuerst das M und dann das D.

Das nächste Beispiel zeigt, daß Sie innerhalb **<#....#>** auch rechnen können. Es nimmt eine Zeit in Sekunden, wandelt sie um und zeigt sie in der Form Stunden : Minuten : Sekunden an.

```
: MINSEC
  ( Dividiert den TOS durch 6Ø, speichert den Rest in
  zwei Ziffern und setzt danach Doppelpunkt)
  ( n - n/6Ø)
  6Ø /MOD SWAP
  Ø # # DROP DROP
  ASCII : HOLD
;

: ZEIT
  ( Anzahl Sekunden - )
  <# MINSEC ( : Sekunden)
  MINSEC ( : Minuten)
  Ø #S ( Stunden)
  #> TYPE
;
```

Ein weiteres Wort im Zusammenhang mit **<#** und **#>** ist **SIGN** (Ganzzahl -). **SIGN** nimmt aus dem stack eine einfachlange Ganzzahl mit Vorzeichen und überträgt ein Minuszeichen in den Zwischenspeicher, wenn sie negativ ist.
Da **#** Zahlen ohne Vorzeichen verwendet, müssen Sie mit **#** den Absolutwert verarbeiten und irgendwann **SIGN** einsetzen. Mit dem nachfolgenden Wort **D->PAD** kann man eine doppeltlange Ganzzahl mit Vorzeichen anzeigen.

```
: D->PAD
  ( doppeltlange Ganzzahl - )
  DUP >R DUP Ø<
  IF
    DNEGATE
  THEN
  <# #S R> SIGN #>
;

: D.
  ( doppeltlange Ganzzahl - )
  D->PAD TYPE
;
```

Wir stellen zunächst das Vorzeichen der Zahl sicher. Es wird im höchstwertigen Bit der Zahl dargestellt (als Ø für positive Zahlen oder Ø, als 1 für negative Zahlen).

D->PAD hat die beiden Teile der Zahl zur Verfügung (geringerwertiger Teil , höherwertiger Teil).

DUP >R sichert das Vorzeichen (als Teil der höherwertigen Hälfte) im RETURN-stack.

Im stack steht jetzt noch die Zahl mit Vorzeichen. Ist sie negativ, wird sie mit DNEGATE umgekehrt. Auch dazu genügt es, die höherwertige Hälfte zu betrachten.

Es folgt die Übertragung der Ziffern in den Zwischenspeicher mit #S, und danach wird das Vorzeichen mit R> SIGN ergänzt.

Jetzt ist das Arbeiten mit doppeltlangen Zahlen schon besser zu erkennen. Versuchen Sie z.B.

```
256 256 U* D.
```

Bekannte Wörter aus der Einfachlängen-Arithmetik können entsprechend auf Doppellänge angepaßt werden. Hier einige Beispiele:

```
: DØ=
  ( d - MERKER)
  OR Ø=
;

: DØ<
  ( d - MERKER)
  SWAP DROP Ø<
;

: DABS
  ( d - Absolutwert von d)
  DUP Ø<
  IF
    DNEGATE
  THEN
;
```

Definieren Sie weitere Wörter. Sie können auch die im Kapitel 15 bereits definierten Wörter zur Behandlung doppeltlanger Zahlen heranziehen.

Eine Umkehrung der formatierten Ausgabe ist das Wort **CONVERT**, das Text in eine doppeltlange ganze Zahl umwandelt. Was damit gemeint ist, soll an den Zeichen "123" gezeigt werden, die wir in eine Zahl mit der Basis 1Ø umwandeln wollen. Wir verwenden dazu einen Akkumulator, dessen Inhalt zu Anfang Null ist. Hier werden die bereits bearbeiteten Ziffern gesammelt. In jedem Folgeschritt multiplizieren wir den Akkumulator mit 1Ø und addieren die neue Ziffer hinzu.

Wir werden im folgenden den Inhalt des Akkumulators in Worten darstellen, damit der Unterschied zwischen dem Text "123" und der resultierenden Zahl klar ersichtlich ist.

Erste Stufe: Akkumulator = Null.

Zweite Stufe: "1" lesen. Akkumulator mit zehn multiplizieren und eins addieren. Ergebnis: eins.

Dritte Stufe: "2" lesen. Akkumulator wird eins * zehn + zwei = zwölf.

Vierte Stufe: Akkumulator wird hundertdreiundzwanzig.

Dies ist tatsächlich die Umkehrung von #S. Sie arbeitet auch in jeder anderen Zahlenbasis.

99

CONVERT beginnt mit einem doppelt langen Akkumulator auf dem stack. Am Anfang des Akkumulators steht die Adresse des Zeichens ein Byte vor dem Text, den Sie umwandeln wollen. **CONVERT** liest die Zeichen einzeln und füllt den Akkumulator auf, bis es ein Zeichen findet, das in der aktuellen Zahlenbasis keine Ziffer ist. Dann stoppt es und legt auf dem stack den neuen Akkumulator und die Adresse dieses Zeichens ab:

(Akkumulator, Adresse des Bytes vor dem Text - neuer Akkumulator, Adresse des Zeichens, das keine Ziffer mehr ist).

Als Beispiel definieren wir ein Wort **CONVERT,**, das bei einer gegebenen Adresse beginnend, eine doppeltlange Zahl einliest und Kommata unterdrückt.

```
: CONVERT,
  ( Adresse - doppeltlange Zahl,Adresse d.letzten Feldes)
  1- Ø Ø ROT
  BEGIN
    CONVERT DUP C@ ASCII , -
  UNTIL
;
```

Zum Abschluß dieses recht "durchwachsenen" Kapitels wollen wir noch beschreiben, wie Gleitkommazahlen erzeugt werden. Eine Gleitkommazahl benutzt zwar auch vier Bytes wie eine doppeltlange Zahl, aber in einer ganz anderen Weise. Jede Gleitkommazahl kann in der Form

.xxxxxxEy

geschrieben werden. Sie besteht aus Dezimalpunkt, 6 Ziffern (von denen die erste nicht Ø ist) und einem Exponenten y. Die Zahl 123.456 wird also in der Form .123456E3; die Zahl Ø.ØØØ1234 in der Form .123ØØE-3 geschrieben.

Drei der vier Bytes werden für die Speicherung der 6 Ziffern benutzt. In einem Byte sind immer zwei Ziffern untergebracht, je eine in vier Bits. Diese Schreibweise heißt Binary Coded Decimal (=Binär Codierte Dezimalzahl), weil nicht die Zahl insgesamt binär verschlüsselt wird, sondern jede Ziffer für sich.

Sie können das sichtbar machen, indem Sie die Zahlenbasis 16 wählen und eingeben:

```
123.456 U. U.
```

Sie erhalten die Anzeige

```
43|12 3456
Exponent ↑|  ↑Binär codierte
Byte          Dezimalzahl
```

Die Zahl 123.456 belegt zwei Plätze im stack. Der niedrigere Platz enthält die Ziffern 3456 (binär codierte Dezimalzahlen), der höhere Platz die Ziffern 12 und das Exponenten-Byte. Es können nur Exponenten zwischen -63 und +63 vorkommen. Sie können binär in 7 Bits gespeichert werden. Aus technischen Gründen wird nicht die übliche Methode für die Speicherung von Zahlen mit Vorzeichen verwendet. Zum echten Exponenten wird immer 64 hinzugezählt, so daß es nur positive Zahlen zwischen 1 und 127 gibt.

100

Das restliche Bit im Exponenten-Byte ist für das Vorzeichen der gesamten Gleitkommazahl reserviert: 1 für negative, Ø für positive Zahlen. Diese Stelle hat auch bei doppeltlangen Zahlen die gleiche Funktion, so daß DØ< auch anstelle von FØ< für Gleitkommazahlen verwendet werden kann.

In der Gleitkommazahl Ø. sind alle 4 Bytes Ø. Deshalb kann DØ= für Gleitkommazahlen ebenso verwendet werden wie FØ=

Zusammenfassung:

Doppelt lange ganze Zahlen

Formatierte Ausgabe

Gleitkommazahlen

FORTH-Wörter D+, D< , DNEGATE, U*, U/MOD, */, <#, #, #S, #> , HOLD, SIGN, CONVERT

Übungen:

1. Schreiben Sie eine Version von GELD, die immer drei Stellen links vom Komma ausgibt und einen oder mehrere Sterne (*) anzeigt, wenn kein dreistelliger DM-Betrag vorhanden ist (sog.Schutzstellung), z.B. DM**1.95

2. Definieren Sie die Doppellängen-Version bereits bekannter Wörter: D-, D=, DMAX, DMIN, und DU<.

   In einigen FORTH-Anwendungen gibt es das Wort D.R
   D.R (d,n -) gibt n Stellen einer doppeltlangen Zahl d aus. Nicht benötigter Platz wird mit Leerstellen aufgefüllt. In unserer Definition D.R verwenden wir das früher schon definierte D->PAD.

```
: D.R
  ( d,n - )
  >R D->PAD
  R> OVER - SPACES TYPE
;
```

   Achtung: Ist die Zahl länger als die angegebenen Stellen, erhalten Sie einen Überlauf!

   Definieren Sie ein Wort S->D mit dem Sie eine einfache Zahl mit Vorzeichen in eine doppelt lange umwandeln können.

 101

KAPITEL 20

IM WÖRTERBUCH

Eine Wortdefinition wird, wie Sie wissen, ins Wörterbuch übernommen. Bisher haben wir drei Arten von Wortdefinitionen kennengelernt: **CONSTANT**, **VARIABLE** und : . Obwohl die drei Arten von Wortdefinitionen sehr unterschiedlich sind, haben sie andererseits doch einiges gemeinsam:

1. Jedes Wort hat einen Namen
2. Jedes Wort enthält Informationen, die das zuvor definierte Wort betreffen. Alle Wörter sind nämlich in einer langen Kette miteinander verbunden, (man sagt auch miteinander verkettet); jedes Wort gibt an, wo das nächste zu finden ist. Die Kette beginnt bei dem zuletzt definierten Wort und verläuft (in der Reihenfolge von **VLIST**) bis zum ältesten Wort. Dort steht ein Kennzeichen, das besagt, daß keine weitere Verbindung mehr besteht.
3. Jedes Wort hat ein 2 Bytes langes Code-Feld, das grundsätzlich angibt, was bei Ausführung des Worts zu tun ist. **CONSTANT** legt zum Beispiel eine Zahl in einen Speicher, : führt weitere FORTH-Wörter aus.
4. Jedes Wort enthält weitere Informationen (seine Parameter), die genauere Aussagen machen, z.B. die Zahl, die **CONSTANT** speichern soll, oder die Wörter, die : ausführt.

Name, Verkettung und Codefeld haben für jede Wortart das gleiche Format. Sie bilden den Anfang (engl.: Header = Vorsatz) des Worts. Jedes Wort hat einen Header.

Die Parameter sind im Format sehr verschieden. Sie reichen von einer einzigen Zahl bis zu einem großen FORTH-Programm.

Die Parameter bilden das Parameter-Feld. Wie sie anzuwenden sind, ist im Codefeld festgelegt.

Diese Festlegungen gelten ganz allgemein. Sie bedeuten, daß sich der Wortschatz nicht nur auf die bekannten FORTH-Wörter :, **CONSTANT**, **VARIABLE** beschränken muß, sondern daß man eigene Wort-Typen erzeugen kann. Die einfachste Methode hierfür bietet das Wort **CREATE**. Es bildet ein Wort, das nur einen Header (Vorsatz), aber kein Parameterfeld hat, und stellt es ins Wörterbuch.

Ein Wort ohne Parameterfeld scheint zunächst recht sinnlos zu sein. Dies ist aber nicht so, denn

- Sie können ein Parameterfeld dafür festlegen (und werden das in der Regel auch tun),
- das Wort legt bei seiner Ausführung die Adresse des Parameterfeldes im stack ab. Sie können mit Hilfe dieser Adresse die Parameter einsetzen, wie Sie wollen.

Normalerweise verwendet man **CREATE** dazu, Variable mit mehr als einer Zahl zu speichern. Man nennt diese Variablen Bereichsvariablen (engl.: arrays). Wollen Sie z.B. Zahlen speichern, die die Anzahl der Tage aller 12 Monate wiedergeben, dann können Sie wie folgt verfahren: Bilden Sie ein Wort **MONATE**, dessen Parameterfeld diese 12 Zahlen in der richtigen Reihenfolge enthält.

102

```
|-----------------------------------------------------------------
|  Header f. MONATE    |31|28|31|3Ø|31|3Ø|31|31|3Ø|31|3Ø|31|
-----------------------------------------------------------------
                        |---Parameter-Feld----------------|
```

Den Header bringen Sie mit

CREATE MONATE

in das Wörterbuch.

Für das Parameterfeld gibt es ein Wort , (Komma). , (n -) nimmt eine Zahl aus dem stack und fügt sie in das Wörterbuch ein. Für die zwölf Zahlen geben Sie ein:

31 , 28 , 31 , 3Ø , 31 , 3Ø ,

31 , 31 , 3Ø , 31 , 3Ø , 31 ,

Das , ist ein FORTH-Wort, kein Satzzeichen. Es muß durch Zwischenraum von den Zahlen getrennt sein. Auch das , nach der letzten Zahl ist notwendig.

Das Wort **MONATE** ist jetzt völlig definiert. Jetzt definieren wir ein Wort **MONAT**, das für die eingegebene Monatsnummer die Zahl der diesem Monat zugeordneten Tage bereitstellt.

```
: MONAT
  ( Monat - Zahl der Tage)
  1- DUP + MONATE +
  @
;
```

Die Definition für **MONAT** wird im Wörterbuch unmittelbar nach der letzten Zahl (31) von **MONATE** gespeichert.

MONATE stellt im stack die Adresse seines Parameterfeldes, d.h. die Adresse der 31 für Januar, bereit. Dazu müssen wir für Januar eine Ø, für Februar eine 2 usw. addieren. Die Monatssummen sind vorher zu verdoppeln, weil die Zahl der Tage jedes Monats in zwei Bytes angegeben ist. **1- DUP +** macht also aus der eingegebenen Monatszahl 1 - 12 eine <u>relative Adresse</u> der gewünschten Zahl (Ø - 22), die dann zu der Adresse des Parameterfeldes addiert wird, welche **MONATE** bereitstellt.

@ holt schließlich die gewünschte Zahl aus dem Wörterbuch.

Wir zeigen eine ähnliche Anwendung für die Abkürzung der Tagesbezeichnungen (Mon,Die,Mit). Dazu entwickeln wir ein Wort **TAGTEXT**, das die notwendigen 21 Zeichen enthält. Da ein Zeichen nur ein Byte benutzt, brauchen wir hierzu **C,** eine Version von , die auf ein Byte zugreift.

C, (Zahl -) fügt wie , eine Zahl ins Wörterbuch ein, aber immer nur in ein Byte.

Da die Eingabe **ASCII M C, ASCII o C,** usw. auf die Dauer sehr stupide wäre, definieren wir ein Hilfswort **FOLGE**, das uns die Definition von **TAGTEXT** erleichtern soll. Es nimmt ein Wort aus dem Eingabebereich und trägt es zeichenweise ins Wörterbuch ein.

```
: FOLGE
  32 WORD COUNT ( count wurde in Kapitel 16 definiert)
  OVER + SWAP
  DO
   I C@ C,
  LOOP
;
```

103

Geben Sie jetzt ein

```
CREATE TAGTEXT FOLGE  MonDieMitDonFreSamSon
```

Wenn Sie **FOLGE** nicht mehr benötigen, können Sie es mit

```
REDEFINE FOLGE
```

durch **TAGTEXT** überschreiben.

Jetzt brauchen Sie noch ein Wort **TAG**, das zu einer eingegebenen Tages.zahl den entsprechenden Text anzeigt

```
: TAG
  ( Tag -)
  1- 3 * TAGTEXT +
  3 TYPE
;
```

Für die Bereitstellung von Parameterfeldern gibt es noch ein anderes Wort, ALLOT (Anzahl Bytes -). Es stellt eine Anzahl von Bytes im Wörterbuch zur Verfügung, wie das auch **C,** und **,** tun, läßt aber im Gegensatz zu diesen die Speicherplätze leer. Es reserviert also nur den Platz.

CREATE ist ein ziemlich einfaches Wort. Es baut nur einen Header auf, hilft Ihnen aber nicht beim Aufbau des Parameterfeldes. Das durch **CREATE** erzeugte Wort schließlich stellt auch lediglich die Adresse des Parameterfeldes im stack ab, das ist alles. Das ist ja ganz schön, wenn Sie Ihr Wort nur einmal brauchen, aber bei mehreren ähnlichen Vorgängen tun Sie damit dieselbe Arbeit mehrmals.

Sie wollen z.B. die Tagesbezeichnung in verschiedenen Sprachen haben: DAY in Englisch, JOUR in Französisch.Für jede Sprache brauchen Sie

1. Die Daten (Tagesbezeichnungen) inklusive einer Methode, sie aufzubauen (was FOLGE tut).

2. Ein Wort, das die Daten dann anzeigt, so wie **TAG**.

Dabei ändern sich von Sprache zu Sprache ja nur die Daten. Wir definieren uns deshalb ein Wort MACHTAGE, das die Bezeichnungen sowohl einbaut als auch verwendet. Dazu können wir definieren

```
MACHTAGE TAG MonDieMitDonFreSamSon
```

MACHTAGE kann Daten aufbauen, **TAG** enthält selbst die Daten, **TAGTEXT** wird nicht mehr gebraucht. Wenn wir **TAG** so verwenden wie vorher, nimmt es Bezug auf **MACHTAGE**.

Wie geht das? Zunächst definieren wir **MACHTAGE**, aber nicht mit **:** sondern mit zwei neuen Wörtern **DEFINER** und **DOES>** . Sie werden immer zusammen benutzt. (Geben Sie vorher noch ein **FORGET TAGTEXT** und FORGET TAG ein. Sie brauchen beide nicht mehr).

```
DEFINER MACHTAGE
  32 WORD 1+ DUP 21 + SWAP
  DO
    I C@ C,
  LOOP
DOES>
  SWAP 1- 3 * +
  3 TYPE
;
```

(Der Abschluß erfolgt mit ; wie in einer Doppelpunktdefinition).

104

Der ganze Vorgang besteht aus zwei Teilen:

Der Definitionsteil geht bis **DOES>** .

Wenn wir sagen

MACHTAGE TAG MonDieMitDonFreSamSon

erzeugt **MACHTAGE** zunächst einen Header für **TAG**, genau wie **CREATE**, wobei das Codefeld anders aussieht, wie wir noch sehen werden.
Dann baut **MACHTAGE** in seinem Definitionsteil das Parameterfeld für **TAG** auf, - es liest in unserem Fall eine Zeichenfolge (engl.:string) aus dem Eingabebereich und fügt 21 Zeichen davon ins Wörterbuch ein, wie das auch das Wort **FOLGE** getan hat. **TAG** ist jetzt richtig definiert. **MACHTAGE** ist zunächst fertig.

Teil 2 von **MACHTAGE**, der Teil von **DOES>** bis ; ist der Aktionsteil. Er wird eingesetzt, wenn wir sagen

3 **TAG**

Wäre **TAG** mit **CREATE** aufgebaut worden, würde nicht viel geschehen - lediglich die Adresse des Parameterfelds würde auf dem stack hinterlegt. Jetzt hingegen kann **TAG** mehr: Mit Hilfe von **MACHTAGE** legt **TAG** zwar immer noch die Adresse auf dem stack (über die eingebene 3) ab, führt aber dann den Aktionsteil von **MACHTAGE** aus. Dieser Teil benutzt die beiden Zahlen, um die Kurzbezeichnung zu suchen und anzuzeigen.

Geben Sie ein

MACHTAGE JOUR lunmarmerjeuvensamdim

MACHTAGE DAY montuewedthufrisatsun

4 **JOUR**

5 **DAY**

Dies ist eine der klügsten Ideen von FORTH, die zu beherrschen sich lohnt!

Beachten Sie bitte, daß **MACHTAGE** nicht eines der bisherigen Wald- und Wiesenworte ist. Es hat vielmehr die Fähigkeit, neue Wörter zu definieren und steht auf einer Ebene mit **CONSTANT**, **VARIABLE** und **CREATE**. **DEFINER** steht sogar noch weiter oben, denn es kann neue Wörter definieren, die ihrerseits wieder neue Wörter definieren können. Im nächsten Kapitel werden wir sehen, wie man damit Erleichterungen schaffen kann, an denen es im FORTH in seiner Grundform fehlt.

Beachten Sie weiter: Wenn Sie ein definierendes Wort (wie **MACHTAGE**) benutzen, und wenn während der Definition eines neuen Wortes (z.B. **TAG**) ein Fehler auftritt, geht die unvollständige Definition im Wörterbuch verloren. Dadurch wird zwar weiter kein Schaden angerichtet, aber es wird Speicherplatz verschwendet. Sie sollten es deshalb mit **FORGET** löschen. In Kapitel 24, Übung 5 zeigen wir Ihnen einen anderen Weg.

Noch eine Anmerkung:
In anderen FORTH-Versionen wird statt

DEFINER Name

geschrieben

: Name **CREATE**

oder

: Name **<BUILDS**

Die Wirkung ist gleich, im ACE ist aber nur **DEFINER** möglich.

Schauen wir uns zum Schluß noch an, wie der Header eines Wortes genau aufgebaut ist:

Zuerst kommt das Namensfeld. Es enthält den Namen und hat für jedes Zeichen ein Byte (maximal 63). Buchstaben werden in Großschreibung dargestellt. Das letzte Zeichen zur Kennzeichnung wird um 128 erhöht (d.h. daß das höchstwertige Bit auf "1" geschaltet wird). Normalerweise bedeutet dieses Zeichen die Umschaltung auf Invers Video, nicht aber hier.

An zweiter Stelle folgen zwei Bytes, das Längenfeld. Es gibt die Gesamtlänge der Wortdefinition, ausgenommen das Namensfeld, an: 7 für den Rest des Headers + Länge des Parameterfeldes. Das Längenfeld wird ausgefüllt, wenn das nächste Wort definiert wird.

Das Dritte ist das zwei Bytes lange Verkettungsfeld. Es ist die Adresse des Namenslängenfeldes des unmittelbar vorher definierten Wortes.

An vierter Stelle folgt ein Byte als Namenslängenfeld, also die Zahl der Zeichen, aus denen der Name besteht. Um aus einem Wort ein Sofortwort zu machen (s.Kapitel 23), kann zu der Länge noch die Zahl 64 addiert werden.

Wort fünf ist das Codefeld, das in zwei Bytes angibt, wie sich das Wort verhalten soll.

Mit dem Wort **FIND** erhalten Sie die Adresse des Codefeldes des angegebenen Wortes.

Zusammenfassung:

Header (Vorsätze) von Wörtern - Namensfelder, Längenfelder, Verkettungsfelder, Namenslängenfelder, Codefelder, Parameterfelder.

FORTH-Wörter: **CREATE** **,** **C,** **ALLOT** **DEFINER** **DOES>**

Übungen:

1. Wenn es **VARIABLE** und **CONSTANT** nicht gäbe, wie könnte man sie mit **DEFINER** definieren? Definieren Sie ähnliche Wörter **2VARIABLE** und **2CONSTANT**, die 4-Bytes-Zahlen speichern können, d.h. entweder Gleitkommazahlen oder Ganzzahlen mit Doppelter Genauigkeit.

2. Definieren Sie ein Wort **KOPF** mit **CREATE** und vergleichen Sie

   ```
   KOPF .
   ```

   mit

   ```
   FIND KOPF .
   ```

   Das erste Wort gibt die Adresse des Parameterfeldes, das zweite die Adresse des Codefeldes an, das 2 Bytes davor liegt.

3. Viele FORTH-Versionen haben ein Wort **'** (ausgesprochen "tick"), das wie **FIND** arbeitet, aber die Adresse des Parameterfeldes ausgibt. Seine Definition auf dem ACE wäre

   ```
   : '
     FIND 2+
   ;
   ```

4. In Kapitel 11 Übung 1 haben wir versprochen, Ihnen eine schnelle Methode der Berechnung von Tonhöhen-Zahlen vorzustellen. Das Problem bestand darin, eine gegebene Tonhöhe mit $(1/2)^{n/12}$ zu multiplizieren, um die Daten eines Halbtons zwischen Ø und 11 zu errechnen. Dazu haben wir sie n-mal mit $(1/2)^{1/12}$ multipliziert. Es ist aber besser, die 12 Zahlen in einem Bereich zu speichern. Jede wird mit 2 Zahlen als Bruch definiert (z.B. 17843/189Ø4 für $(1/2)^{1/12}$).

Definieren Sie

```
CREATE SKALA 1 , 1 , 17843 , 189Ø4 ,
26547 , 29798 , 16585 , 19723 ,
4813 , 6Ø64 , 5Ø89 , 6793 ,
196Ø1 , 2772Ø , 6793 , 1Ø178 ,
3Ø32 , 4813 , 5377 , 9Ø43 ,
14899 , 26547 , 9452 , 17843 ,
```

(zusammen 24 Zahlen).

Die neue Version von SEMI heißt jetzt

```
: SEMI
  ( Halbtöne über kleinem c - Tonhöhe)
  36 + ( Halbtöne über dem tiefen C)
  12 /MOD SWAP ( Zahl Oktaven, Zahl Halbtöne)
  3822 SWAP
  DUP + DUP + SKALA +
  DUP @ SWAP 2+ @ */
  SWAP ?DUP
  IF
    ( Division durch 2 für jede Oktave)
    Ø
    DO
     2 /
    LOOP
  THEN
;
```

5. Hier folgt noch eine hübsche Prüfmöglichkeit für TAG, JOUR und DAY unter Verwendung von FIND und EXECUTE

```
: TEST
  FIND 8 1
  DO
    I OVER EXECUTE SPACE
  LOOP
  DROP
;
```

Nun geben Sie ein

```
TEST TAG
```

107

KAPITEL 21

ZEICHENKETTEN UND BEREICHE

Zwei Anwendungsbeispiele, die **DEFINER** benutzen:

Strings (=Zeichenketten) sind Aneinanderreihungen von Zeichen, die als eine Einheit behandelt werden.

Bereiche (=Arrays) sind Variable, die mehr als eine Zahl speichern können.

In vielen Computersprachen sind Strings und Arrays bereits eingebaut. Bei FORTH ist dies normalerweise nicht so, aber Sie können diese Hilfsmittel selbst definieren oder als fertiges Produkt auf Kassette kaufen.

Strings

Ein String ist eine Kette (oder Folge) von Zeichen. Es ist nicht das gleiche wie ein Wort, weil ein Wort im Gegensatz zum String eine vordefinierte Funktion hat. Mit FORTH können Sie Strings mit ."lediglich anzeigen, aber in vielen Computersprachen können Strings fast wie Zahlen verarbeitet werden. Wir zeigen Ihnen hierfür einige Beispiele.

Zunächst suchen wir eine Möglichkeit, Strings auf dem stack abzulegen, allerdings geht das nicht direkt. Während für eine Zahl grundsätzlich feststeht, wieviel Platz sie im stack braucht (zwei Bytes für eine ganze Zahl, vier Bytes für eine doppelt lange Zahl), kann ein String jede Länge bis maximal 255 Bytes haben. Deshalb legen wir den String irgendwo im Speicher ab und notieren seine Adresse und Länge im stack. Das ist wiederum genau die Form, in der **TYPE** arbeitet, so daß wir den String schon anzeigen können. Im nächsten Schritt legen wir ein Verfahren fest, mit dem Strings anstelle von Zahlen gespeichert werden können.

Dazu definieren wir ein Wort **STRING**. Sein Definitionsteil erzeugt ein Parameterfeld, das zunächst ein Byte für die Länge enthält (der String kann deshalb auch nicht länger als 255 Bytes sein), und dann den String selbst, der aus dem Eingabebereich gelesen wurde. Der Aktionsteil setzt die Adresse des Parameterfeldes in Adresse und Länge des String um und legt sie in der üblichen Form auf dem stack ab. Wir verwenden das Wort **COUNT** aus Kapitel 16:

```
DEFINER STRING
  ASCII " WORD COUNT DUP C,
  OVER + SWAP
  DO
   I C@ C,
  LOOP
DOES>
  COUNT
;
```

Definieren Sie nun einen String, indem Sie alles in einem Vorgang eintasten:

```
STRING SPRUCH WER DEN SCHADEN HAT, BRAUCHT FUER DEN SPOTT
NICHT ZU SORGEN"
```

Da wir mit **ASCII " WORD** das Anführungszeichen " als Begrenzer definiert haben, können wir im String Leerstellen verwenden.
Mit

```
SPRUCH TYPE
```

können Sie das Sprichwort wieder anzeigen.

108

Aus einem String kann man Teile herausnehmen (sogenannte Substrings). Man nennt diese Methode slicing (=schneiden). Sie müssen dazu nur angeben, wo der Substring beginnen und enden soll, und natürlich ein Wort SLICE definieren. Das Wort soll so beschaffen sein, daß es aufgrund der Angabe

```
SPRUCH 35 5Ø SLICE TYPE
```

den Text "DEN SPOTT NICHT" anzeigt.

SLICE hat vier Operanden: Adresse und Länge des großen Strings, sowie Anfang und Ende des innerhalb liegenden Substrings. Da hieraus die übliche Formatangabe für String resultiert, können Sie mit dem Substring das Gleiche tun wie mit dem ursprünglichen String. Sie können ihn sogar noch einmal zerschneiden.

```
: SLICE
  ( Adresse, Länge, Anfang, Ende - Adresse, Länge)
  SWAP 1 MAX 3 PICK MIN 1-
  ( Adresse, Länge, Ende, Anfang-1)
  SWAP ROT MIN OVER MAX
  ( Adresse,Anfang-1,Ende)
  OVER - ROT ROT + SWAP
;
```

Diese Definition achtet darauf, daß der Anfang nicht zu niedrig, das Ende nicht zu hoch, oder das Ende nicht niedriger ist als der Anfang.

Man kann Strings miteinander vergleichen. Zwei Strings sind offensichtlich dann gleich, wenn sie dieselben Zeichen in derselben Reihenfolge haben. Etwas schwieriger ist es, Strings in eine alphabetische Ordnung zu bringen. Wir gehen davon aus, daß ein String "kleiner" ist als der andere, wenn er im Alphabet zuerst kommt: "Hund" ist kleiner als "Katze", "Drei" ist kleiner als "Zwei". Wir verwenden dazu die Zeichen < (kleiner als) und > (größer als), die wir in Verbindung mit Zahlen schon kennen.

Die Ordnungsregeln sind hier etwas anders. Sie basieren auf dem ASCII Code der Zeichen. Um zwei Strings zu vergleichen, prüfen Sie Zeichen für Zeichen bis zu einer Stelle, an der Sie voneinander abweichen, und vergleichen diese beiden Zeichen miteinander. Dieses Verfahren stellt sicher, daß "Buchbinder" vor "Buchmacher" kommt. Da aber die ASCII Codes der kleinen Buchstaben größer sind als die der Großbuchstaben, ist überraschenderweise "Zoo" kleiner als "affe", und "FORTH" kleiner als "Forth".

Als String-Versionen von =, < und > definieren wir die Wörter **$=**, **$<** und **$>**.

```
: Ø<>
  ( n - MERKER)
  ( prüft n auf Nicht-Null)
  Ø= Ø=
;
: +COUNT
  ( verwendet in CHECK)
  4 ROLL 1+ 4 ROLL 1-
;
```

Fortsetzung Folgeseite

```
: CHECK
  ( Adresse 1, Länge 1, Adresse 2, Länge 2 - Adresse 3,
  Länge 3, Adresse 4, Länge 4)
  ( Gleicht Adressen und Längen zweier Strings aneinender
  an, um übereinstimmende Anfangszeichen zu übergehen)
  BEGIN
    3 PICK Ø<> OVER Ø<> AND
    5 PICK C@ 4 PICK C@ = AND
  WHILE
    ( noch kein String fertig und noch Übereinstimmung)
    +COUNT +COUNT
  REPEAT
;

: <DROP
  ( a,b - b)
  SWAP DROP
;

: $=
  ( A1, L1, A2, L2 - MERKER)
  CHECK ( die Strings sind jetzt gleich, wenn beide Längen
        Ø sind)
  <DROP OR <DROP ( Adressen fallen weg, Längen mit OR
zusammengekoppelt)
  Ø=
;

: $<
  ( A1, L1, A2, L2 - MERKER)
  CHECK ROT ( A3, A4, L4, L3)
  OVER Ø<> OVER Ø<> AND
  IF
    ( noch kein String zu Ende, verschiedene Zeichen ver-
    gleichen)
    DROP DROP C@ SWAP C@ >
  ELSE
    ( Ein String startet den anderen)
    > <DROP <DROP
  THEN
;

: $>
  ( A1, L1, A2, L2 - MERKER)
  4 ROLL 4 ROLL $<
;
```

Das entscheidende Wort ist **CHECK**, das beide Strings prüft, bis es ein Zeichen findet, in dem beide Strings sich voneinander unterschieden. Auf "Buchbinder" und "Buchmacher" angewendet, endet **CHECK** mit "binder" und "macher".

110

Bereiche

Ein Bereich (engl.array) ist eine Variable, die mehr als eine Zahl aufnehmen kann, wie zum Beispiel **MONATE**. Diese Zahlen heißen Elemente des Bereichs, **MONATE** hat also zwölf Elemente, nämlich 31, 28, 31 usw. Um ein besonderes Element anzusprechen, nehmen Sie seine Position im Bereich, seinen Index. In **MONATE** sind die Indices für Januar = 1, für Februar = 2 usw.

Da wir **MONATE** mit **CREATE** definiert haben, mußten wir auch **MONAT** beschreiben, um den Index in die Adresse eines Elements umzuwandeln. Mit **DEFINER** kann man ein Wort **BEREICH** definieren, das man wiederum zur Definition von **MONATE** verwenden kann. **MONATE** kann dann sowohl die Elemente speichern als auch die Indices verarbeiten (eine schöne Übung).

Eine kompliziertere Bereichsdefinition benutzt zwei Indices für jedes Element: Der Bereich ist zweidimensional. Die Elemente sind mit Zahlen in einer Tabelle zu vergleichen, die man mit einem Index für die Zeile und einem zweiten für die Spalte ansprechen kann (vgl.S.72).

Mit den nachfolgenden Wörtern können Sie zweidimensionale Bereiche definieren. Es sind nur Beispiele, mann kann noch eine Menge ergänzen. Ist Ihr Speicher nicht groß genug, löschen Sie vorher die Wörter zum Thema String.

```
: 2*
  ( n - 2*n)
  DUP +
;

: ZEILE
  ." Zeilenfehler"
  CR
;

: SPALTE
  ." Spaltenfehler"
  CR
;

: NACHRICHT
  ." Bitte FORGET dieses Wort."
  CR ABORT
;

: ZEILE?
  IF
    ZEILE ABORT
  THEN
;

: SPALTE?
  IF
    SPALTE ABORT
  THEN
;
```

Fortsetzung Folgeseite

```
DEFINER 2-D
( Zeilenzahl, Spaltenzahl)
DUP 1- Ø<
IF
  ( Zahl d.Spalten ist Ø oder kleiner)
  SPALTE NACHRICHT
THEN
OVER DUP 1- Ø<
IF
  ( Zahl d.Zeilen ist Ø oder kleiner)
  ZEILE NACHRICHT
THEN
C, DUP C, * 2* ALLOT
DOES>
  ( Zeile, Spalte, Bereichsadresse - Elementadresse)
  ROT ROT 3 PICK ( Adresse, Zeile, Spalte, Adresse)
  C@ 3 PICK DUP 1-
  Ø< ZEILE? < ZEILE? ( Fehlermeldung, wenn falsche
  Zeile)
  DUP 1- Ø< SPALTE?
  3 PICK 1+ C@ DUP
  3 PICK < SPALTE? ( Fehlermeldung, wenn falsche
  Spalte)
  ROT 1- * + 2* +
;
```

2-D setzen Sie ein, indem Sie z.B. sagen

```
8 8 2-D SCHACH
```

oder

```
1 12 2-D MONATE
```

und dann Ihre Elemente einfüllen. Die Elemente werden wie Variable mit @ und ! verarbeitet. Um z.B. ein Element nach Zeile 2, Spalte 5 in **SCHACH** zu laden, geben Sie ein

```
1 2 5 SCHACH !
  |--------------|
   Legt  Element fest
```

und mit

```
2 5 SCHACH @
```

holen Sie es wieder in den stack.

112

Zusammenfassung:

Strings, Substrings, Vergleich von Strings,
Arrays (Bereiche), Elemente, Indices, Dimensionen.

Übungen:

1. Noch ein paar Verbesserungen zu Strings:

   - Zuordnung: Die Zeichen eines Strings sollen in einen zweiten übertragen werden. Adressen und Längen beider Strings sind im stack vorhanden. Ist der erste String zu lang, soll das Ende abgeschnitten werden, ist er kürzer, sind die übrigen Bytes mit Leerstellen aufzufüllen.
   - Ein Wort, das nur die Länge eines Strings auf dem Stack behält (und die Adresse wegwirft).
   - Ein Wort, das den ASCII-Code des ersten Buchstabens im stack ablegt.

2. Weitere Möglichkeiten für Bereiche:

   - Wenn die Adresse eines Elements errechnet wird (z.B. durch ein Programm), können die Indices geprüft werden, um zu vermeiden, daß sie zu groß oder zu klein werden, wie z.B. in **2-D**. Andererseits wird ein Programm natürlich schneller, wenn man solche Prüfungen wegläßt.
   - Wir haben für einen Bereich Obergrenzen definiert, um festzulegen, wie groß er werden darf. Die Untergrenzen müssen natürlich nicht automatisch bei 1 liegen. Sie können ebenfalls frei definiert werden.
   - Man kann drei- oder mehrdimensionale Bereiche definieren. Die Zahl der Dimensionen entspricht der Zahl der Indices, die zu einem Element gehören. Mit einem Wort **DIM** (statt **2-D**) wird dann erst die Zahl der Dimensionen vom stack gelesen und in das Parameterfeld gestellt. Danach folgen die Grenzen selbst.
   - Ein Bereich (array) muß nicht auf zwei Bytes lange Zahlen für Eintragungen beschränkt sein. Man könnte auch Bereiche für Eintragungen von einem oder vier Bytes definieren.
   - Das Definitionswort könnte alle Elemente auf Ø setzen, wenn der Bereich definiert wird.
   - Ein Konstantenbereich enthält feste Elemente. Diese müssen bei der Definition des Bereichs bereits im stack vorhanden sein.

   Diese Anforderungen sind zum Teil recht schwierig zu realisieren. Versuchen Sie Ihr Glück.

3. Zwei Wörter, die man oft in FORTH findet, nicht aber im ACE FORTH sind **MOVE** und **CMOVE**. Für die Bearbeitung von Strings sind sie sehr nützlich. Sie kopieren Information von einem Speicherplatz zu einem anderen.

   MOVE (Adresse 1, Adresse n -) kopiert den Inhalt von n 2-Bytes langen Stellen ab Adresse 1 in den Speicher ab Adresse 2.

 113

CMOVE (Adresse 1, Adresse 2, n -) ist ähnlich. Es überträgt aber Einzelbytes statt 2-Bytes lange Stellen.

Ist n in **MOVE** oder **CMOVE** negativ oder Ø, geschieht nichts. Schreiben Sie Ihre eigenen Definitionen.

Ein Problem kann auftreten, wenn der Bereich, aus dem gelesen wird, sich mit dem Block überschneidet, in dem geschrieben werden soll: In diesem Fall ist sorgfältig darauf zu achten, von welcher Seite des Blocks Sie zu kopieren beginnen.

Ein verwandtes Wort (wenn auch im ACE nicht vorhanden) ist FILL (Adresse, n, Byte -), das den Speicher ab einer gegebenen Adresse mit n Kopien eines gegebenen Bytes füllt.

 114

KAPITEL 22

MEHRERE WÖRTERBÜCHER

Im FORTH kann man mehrere Wörterbücher nebeneinander aufbauen. Jedes Wörterbuch hat seinen eigenen Namen. Anfangs, nach dem Einschalten des ACE, gibt es nur **FORTH**. Wenn Sie daneben weitere Wörterbücher festgelegt haben und ein Wort suchen, dann beschränkt sich die Suche auf das gerade aktive Wörterbuch. Sie können also z.B. Wörter mit gleichem Namen, aber unterschiedlicher Bedeutung haben, je nachdem, welches Wörterbuch aktiv ist.

Nehmen Sie an, Sie wollten einen "Spickzettel" für Geschichtszahlen anlegen. Sie können das natürlich so tun, daß die Zahl, die Sie sich merken wollen, gleichzeitig das definierte Wort ist

```
: 333
  ." Bei Issus Keilerei"
;
```

ein Datum, das jeder kennt, der in Geschichte aufgepaßt hat.

Es ist zwar ganz schön, daß nun **333** immer den Text " Bei Issus Keilerei" aufruft. Gleichzeitig aber ist 333 als normale Rechengröße nicht mehr zu gebrauchen. Wegen der vielen Geschichtszahlen wäre der ACE für das normale Rechnen bald unbrauchbar, darum **FORGET 333**.

Es ist viel sicherer, solche Spezialitäten in einem eigenen Wörterbuch unterzubringen. Wir nennen es **HISTORIE** und eröffnen es mit

```
VOCABULARY HISTORIE
```

Es geschehen zwei Dinge:

1. Ein Leitwort ("Wörterbuch-Definitions-Wort") **HISTORIE** wird definiert, dessen Aufgabe es ist, **HISTORIE** zu aktivieren. Dieses Wort ist Bestandteil des **FORTH**-Wörterbuches.
2. Ein neues Wörterbuch **HISTORIE** wird eröffnet. Bisher enthält dieses Wörterbuch noch kein eigenes Wort, nicht einmal seinen eigenen Namen. Aber alle Wörter des **FORTH**-Wörterbuches sind bereits darin enthalten.

FORTH ist der "Vater" von **HISTORIE**, weil das Leitwort von **HISTORIE** in **FORTH** eingefügt wurde. Wird in einem Wörterbuch ein Wort gesucht, aber nicht gefunden, dann erfolgt die weitere Suche im "Vater"-Wörterbuch.

Nehmen Sie **VLIST**. Sie sehen die **FORTH**-Wörter, wie gewöhnlich. Sie beginnen mit dem Leitwort **HISTORIE**.

Tasten Sie jetzt **HISTORIE** ein. Damit wird **HISTORIE** zum Context-Wörterbuch, jenes, in das wir Wörter eingeben wollen. Mit **VLIST** können wir jetzt das Wörterbuch **HISTORIE** mit allen darin vorkommenden Wörtern anzeigen. Danach folgen die Wörter seines "Vaters", nämlich **FORTH**. Bisher gibt es dabei natürlich noch keinen Unterschied.

Wir wollen jetzt neue Wörter in das Wörterbuch definieren. Durch die Eingabe seines Namens haben wir **HISTORIE** bereits zum Context-Wörterbuch gemacht. Damit können aber nur Wörter aufgerufen werden. Neue Wörter werden aber immer ins laufende (Current) Wörterbuch eingetragen, und das ist bisher noch **FORTH**. Erst mit dem Wort **DEFINITIONS** wird das Context-Wörterbuch auch das Current-Wörterbuch.

115

Nachdem **HISTORIE** jetzt Context-und Current-Wörterbuch ist, können wir unsere Zahlen eingeben.

```
: 333
  ." Bei Issus Keilerei"
;

: 1492
  ." Entdeckung Amerikas"
;

: 1836
  ." Erste Eisenbahn Nuernberg-Fuerth"
;
```

Mit **VLIST** sehen Sie alle bisher definierten Wörter, auch die des FORTH-Wörterbuchs. Die Eingabe FORTH VLIST dagegen zeigt nur die **FORTH** Wörter, nicht unsere Geschichtszahlen. Im **FORTH** haben auch die Zahlen 333, 1492 und 1836 ihre ursprüngliche Bedeutung als Rechengrößen wieder.

Wie arbeiten Wörterbücher? In Kapitel 20 haben wir festgestellt, daß im Header eines Wortes ein Verbindungsglied enthalten ist, das auf das vorher definierte Wort hinweist. Das stimmt aber nicht ganz: Der Hinweis erfolgt vielmehr auf das zuletzt im laufenden (current) Wörterbuch definierte Wort, und es können inzwischen durchaus Wörter in einem anderen Wörterbuch definiert worden sein. In unserem Beispiel hängt 1836 an 1492, und 1492 an 333. Diese Kette bildet das Wörterbuch HISTORIE. Durch eine besondere Art der Verbindung wird das Wörterbuch **HISTORIE** an seinen Vater **FORTH** angehängt.

Wenn jetzt mit **FORTH VOCABULARY** wieder **FORTH** current wird, dann schließt das nächste hier definierte Wort an HISTORIE an, denn dies war ja das zuletzt in **FORTH** definierte Wort. Auf diese Weise können sich Wörterbücher überlappen.

Fassen wir noch einmal zusammen:

1. Ursprünglich existiert nur das Wörterbuch **FORTH**.
2. Es können zwei Wörterbücher zugleich aktiv sein. Das Context-Wörterbuch wird für die Suche nach Wörtern verwendet, das Current-Wörterbuch für die Definition neuer Wörter.
3. **VOCABULARY** definiert ein neues Wörterbuch und das dazugehörige Leitwort. Das Leitwort ist Bestandteil seines "Vater"- (des Current-) Wörterbuches.
4. Mit **DEFINITIONS** wird das Context- auch zum Current-Wörterbuch.
5. Sucht das System nach einem Wort, beginnt es die Suche im Context-Wörterbuch und setzt sie bei Bedarf im "Vater"-Wörterbuch fort. Die Suche endet letztlich im **FORTH**, denn **FORTH** ist Adam und Eva aller Wörterbücher.

Das Parameterfeld eines Leitworts hat folgenden Aufbau:

Zwei Bytes enthalten die Adresse des Namenslängenfeldes des neuesten Wortes im Wörterbuch. Damit kann die Suche nach einem Wort begonnen werden; diese Adresse ist aber gleichzeitig auch die Verbindungsadresse für ein neues Wort.
Ein Byte enthält immer 0. Dieses Byte dient zur Verkettung der Wörterbücher untereinander. Hat ein Wörterbuch einen "Nachkommen", ist dessen erstes (=ältestes) Wort hier verkettet.

116

Zwei Bytes enthalten die Adresse der entsprechenden beiden Bytes in einem anderen Wörterbuch (die Bytes heißen "Wörterbuch-Verkettung") Folgt man dieser Kette, beginnend im neusten Wörterbuch, findet man jeweils das vorhergehende Wörterbuch. Im **FORTH** sind die beiden Bytes dann Ø.

Drei Variable haben mit Wörterbüchern zu tun. Die beiden ersten, **CONTEXT** und **CURRENT**, enthalten jeweils die Parameterfeld-Adressen des Context-bezw. Current-Wörterbuches.

Die dritte Variable hat keinen Namen, aber die Adresse 15413. Ihr Wert ist die Adresse der beiden Bytes, die die Wörterbuch-Verkettung im neusten Wörterbuch enthalten. Von hier aus kann man alle Wörterbücher der Reihe nach ansprechen.

Sie müssen vorsichtig sein mit **FORGET** und **LOAD**.

Eine Regel für **FORGET**: Vergessen Sie immer nur Wörter aus einem Wörterbuch auf einmal. Kommen Sie mit **FORGET** nämlich über die Grenze eines Wörterbuches, geht dessen Leitwort mit verloren. Die Wörterbuch-Verkettungen stimmen nicht mehr.
Auch mit **LOAD** geht die Verkettung verloren.

Zusammenfassung:

Wörterbücher - Context und Current

Leitwörter, Verkettungen

FORTH-Wörter **FORTH**, **VOCABULARY**, **DEFINITIONS**, **CONTEXT**, **CURRENT**.

KAPITEL 23

INNERHALB DER DOPPELPUNKT-DEFINITION

In diesem Kapitel wird erklärt, wie wir mehr Einfluß auf Doppelpunkt-Definitionen bekommen können. Diese Aussagen treffen auch für Definitionen mit **DEFINER** oder (wie wir noch sehen werden) mit **COMPILER** zu.

Wird ein Wort innerhalb einer Doppelpunkt-Definition aus dem Eingabebereich gelesen, dann erfolgt keine sofortige Ausführung, sondern es wird als Teil der Definition abgespeichert. Wir befinden uns im Kompiler-Modus. Das heißt, daß ein Wort als Bestandteil der Definition in Maschinensprache übersetzt und im Wörterbuch gespeichert wird. (engl.:compile).

Normalerweise befindet sich der Computer im Interpreter-Modus. Ein eingegebenes Wort wird übersetzt und sofort ausgeführt (engl.:interpret)

Sie können aber aus einer Doppelpunkt-Definition zeitweise in den Interpreter-Modus zurückschalten, wenn Sie die Wörter [und] benutzen.

[schaltet in den Interpreter-Modus

] schaltet in den Kompiler-Modus.

Nur im Interpreter-Modus gibt der Computer OK aus. So erinnert er Sie daran, in welchem Modus er gerade arbeitet.

Sie verwenden z.B. in einer Definition zwei Zahlen und wollen sie eintasten, indem Sie verschiedene Zahlenbasen benutzen. Während Sie die Basis verändern, gehen Sie mit [und] in den Interpreter-Modus. Nehmen wir das Wort TAB aus Kapitel 12, Übung 4 dazu. Hier kommt die Zahl 31 vor, die der binären 11111 entspricht. Sie können TAB auch so eintasten

```
: TAB
  ( Tabellenende)
  15388 @ -
  [ 2 BASE C! ]
  11111
  [ DECIMAL ]
  AND SPACES
;
```

Jetzt geben Sie ein

```
LIST TAB
```

Im Listing finden Sie nichts mehr von dem, was Sie im Interpreter-Modus eingegeben haben. 11111 ist in 31 umgewandelt worden.

Das Wort [ist also eine Besonderheit, denn obwohl der Computer im Kompiler-Modus ist, wenn er [antrifft, führt er es aus, anstatt es ins Wörterbuch zu speichern. Deshalb nennt man [auch ein Sofort-Wort (eng.:immediate), ein Wort also, das immer sofort ausgeführt wird, auch im Kompiler-Modus.

Sie können Ihre eigenen Sofort-Wörter sehr leicht definieren. Dies geschieht zunächst wie üblich. Danach aber führen Sie das Wort **IMMEDIATE** aus. **IMMEDIATE** verwandelt das neueste Wort im Wörterbuch in ein Sofort-Wort.

118

Wenn Sie vorhatten, mehrere Wortdefinitionen in Binär- und Dezimal-Schreibweise zu verwenden, könnte es nützlich sein zu definieren

```
: BAS2
  2 BASE C!
;

IMMEDIATE

: BAS1Ø
  DECIMAL
;

IMMEDIATE
```

Dann ist es möglich, **TAB** so zu schreiben

```
: TAB
  ( Tabellenende - )
  15388 @ -
  BAS2 11111 BAS1Ø
  AND SPACES
;
```

Wiederum erscheinen **BAS2** und **BAS1Ø** nicht im Listing.

Eine Anwendungsmöglichkeit von [und] ist, daß Sie Rechnungen während einer Wortdefinition ausführen können. Darüber hinaus kann das Ergebnis dieser Rechnung mit dem Sofort-Wort **LITERAL** in die Definition kompiliert werden, als ob Sie es direkt eingegeben hätten.

Wollen Sie den Bildschirm mit Punkten füllen, müssen Sie 23*32 Punkte eingeben. Vor einem Computer zu sitzen und manuell 23*32 ausrechnen zu sollen, ist schon eine Zumutung. Mit **LITERAL** können Sie direkt definieren

```
: PUNKTE
  CR
  [ 23 32 * ] LITERAL ( 23 32 *)
  Ø
  DO
  ." ."
  LOOP
;
```

Der Kommentar (23 32 *) scheint zunächst überflüssig. Im Listing finden Sie aber keine Spur mehr von **LITERAL**, der Kommentar wird Sie wieder daran erinnern, was vorher gewesen ist.

Wir haben bisher Wörter kennengelernt, die im Kompiler-Modus für die spätere Ausführung gespeichert werden, und Wörter, die sofort ausgeführt werden. Es gibt nun einige Wörter, die Beides können müssen.

; hat eine sofortige Wirkung, die sofort in den Interpreter-Modus zurückführt. Es muß aber auch kompiliert werden, weil es das Ende der später auszuführenden Definition markiert. Es hat eine Kompilierzeit-Wirkung (" In dieser Definition gibt es nichts mehr zu kompilieren"), und eine Laufzeit-Wirkung (" Die Definition ist abgeschlossen, es gibt nichts mehr zu tun").

Auch **LITERAL** hat eine solche Doppelfunktion. Während der Kompilierung wird die Zahl erarbeitet und in die Definition eingebaut, während des echten Ablaufs wird die kompilierte Zahl in den stack gebracht und verarbeitet.

119

Auch **IF**, **THEN** und **DO** arbeiten ähnlich. Während der Kompilierung wird die Struktur des Worts erarbeitet, während der Ausführung werden die notwendigen Sprünge über die Programmabschnitte ausgeführt. Solche Wörter heißen Kompiler-Wörter, weil sie Dinge ins Wörterbuch kompilieren.

Mit dem Wort **COMPILER** können Sie eigene Kompiler-Wörter definieren. Bevor wir aber erklären, was **COMPILER** tut, müssen wir uns ein wenig näher mit den Besonderheiten des Kompilierens befassen.

Jedes FORTH-Wort hat, wie wir aus Kapitel 20 wissen, ein Codefeld im Header. Dessen Adresse ist die Codefeld-Adresse, die das Wort **FIND** auf dem stack ablegt. Wird das Wort in die Definition eines anderen Wortes kompiliert, geschieht nichts weiter, als daß seine Codefeld- Adresse ins Wörterbuch übernommen wird (dazu wird ; benötigt). Deshalb heißt die Codefeld-Adresse auch oft Kompilier-Adresse.

Ein Wort, das die Kompilier-Adresse verwendet, ist **EXECUTE** (Kompilieradresse -). Es nimmt eine Kompilieradresse vom stack und führt das Wort aus, z.B.

```
FIND DUP EXECUTE
```

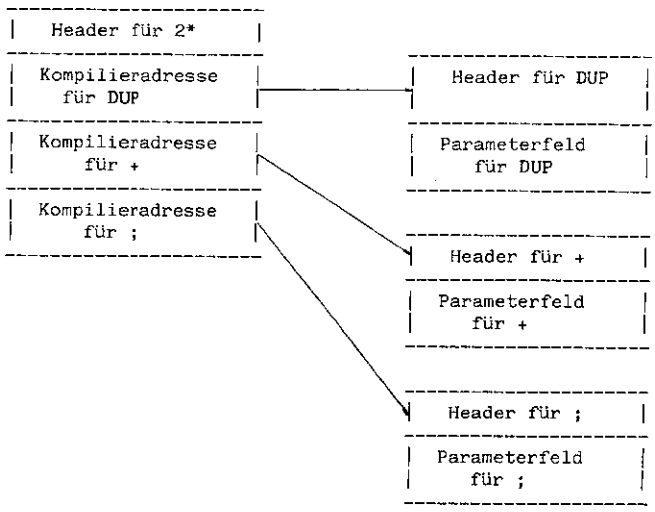
tut das Gleiche wie **DUP**.

Eine einfache Doppelpunkt-Definition enthält in ihrem Parameterfeld eine ganze Liste von Kompilieradressen. Um das Wort auszuführen, nimmt ACE die erste dieser Adressen, sucht das zugehörige Wort und führt dieses aus, kehrt dann zur nächsten Kompileradresse zurück und so weiter. Wenn er den ; erreicht, weiß er, daß das Wort ausgeführt ist.

Wenn Sie z.B. definieren

```
: 2*
  DUP +
;
```

dann hat die Definition für **2***



(**DUP**, + und ; sind im ROM gespeichert und ihre Parameterfelder sind sehr unterschiedlich).

Diese Form ist die einfachste. Oftmals aber müssen hinter der Kompilieradresse noch weitere Informationen angeführt werden. **LITERAL** kompiliert z.B. eine Zahl in die Definition. Diese Zahl wird unmittelbar hinter der Kompilierungsadresse von **LITERAL** gespeichert. Genauso werden auch direkt eingetastete Zahlen gespeichert.

Diese Zusatzinformation heißt Operandenfeld. Kompilieradresse und Operandenfeld zusammen heißen ein kompiliertes Wort.

Ein weiteres Beispiel ist **IF**. Sein Operandenfeld hat zwei Bytes, die aussagen, wie weit zu springen ist, um über den Programmabschnitt **IF...ELSE** hinwegzukommen, wenn die Bedingung falsch ist.

Ein eigenes kompiliertes Wort erzeugen Sie, indem Sie festlegen

1. Wie das Operandenfeld aussieht, und

2. wie das kompilierte Wort ausgeführt wird.

Die Festlegung erfolgt mit den zwei Wörtern **COMPILER** und **RUNS>**. Sie treten immer gemeinsam auf wie **DEFINER** und **DOES>**.

Das Wort **2LITERAL** wirkt wie **LITERAL**, jedoch auf vier Bytes statt auf zwei, es kann also auf Gleitkomma- oder doppelt lange Ganzzahlen angewandt werden

```
: 4 COMPILER  2LITERAL
    SWAP , ,
  RUNS
    DUP @ SWAP 2+ @
;
```

In der Kompilierzeit werden zwei Eintragungen vom stack genommen und mit , , ins Wörterbuch eingefügt. Dies geschieht vor **RUNS>**.
Die Laufzeit-Aktivität ist der Teil nach **RUNS>**. Sie hinterläßt die Adresse des Operandenfeldes auf dem stack. Damit kann man die dort gespeicherte vier Byte lange Zahl holen.

Die 4 vor **COMPILER** sagt aus, wieviel Bytes in das Operandenfeld gebracht werden sollen. Wenn diese Zahl variieren kann, verwenden Sie -1 **COMPILER**, und während der Kompilierzeit werden die tasächlich benötigten Längen in die Bytes eingesetzt. So arbeiten z.B. (und ."

Um Ihnen zu zeigen, wie **2LITERAL** anzuwenden ist, wollen wir ein Wort schreiben, das 1/7 zu einem im stack gespeicherten Wort addieren soll

```
: 1/7+
  1. 7. F/ F+
;
```

Das Wort wird natürlich viel schneller verarbeitet, wenn Sie statt **1/7+** seinen Dezimalwert Ø.142857 eingeben

```
: 1/7+
  Ø.142857 F+
;
```

121

Da es aber lästig wäre, diese Zahl vorab auszurechnen, können Sie einfach eingeben

```
: 1/7+
  [ 1. 7. F/ ] ( 1/7)
  2LITERAL F+
;
```

In der Kompilierzeit wird in die Definition von **1/7+** ein kompiliertes Wort eingesetzt: Zunächst die eigene Kompilieradresse und dann das Operandenfeld mit der Gleitkommazahl. Während der Laufzeit wird die Zahl aus dem Operandenfeld auf den stack kopiert.

Seien Sie vorsichtig, wenn Sie **1/7+** editieren. Der Listing-Ablauf weiß nichts mit dem Operandenfeld anzufangen, er ignoriert es einfach. Sie sehen nur

```
: 1/7+
  ( 1/7)
  2LITERAL F+
;
```

Der einzige Hinweis auf das Operandenfeld besteht im Kommentar. Beim Editieren müssen Sie den Rechenvorgang für 1/7 komplett neu eingeben.

Nützlich ist auch das Wort **HERE** (- Adresse). Es legt auf dem stack die nächste freie Adresse des Wörterbuchs ab. An diese Stelle wird das erste Byte eines neuen Wortes gespeichert. Sie können damit prüfen, wie voll der Speicher inzwischen geworden ist.

Zusammenfassung:

Kompiler- und Interpreter-Modus

Sofort-Worte, Kompilierworte

Kompilierte Wörter: Kompilieradresse und Operandenfeld

FORTH-Wörter [,], **IMMEDIATE**, **EXECUTE**, **COMPILER**, **RUNS>**, **HERE**

Übungen:

EXECUTE kann zusammen mit einer Kompilier-Adresse als gespeicherter Variablen eingesetzt werden. Ein Beispiel: Sie haben zwei Methoden, mit unwillkommenen Anfragen fertig zu werden

```
: GROB
  CR ." LASSEN SIE MICH IN RUHE"
  CR ." UND HAUEN SIE AB"
  CR ." SIE DUMMKOPF"
;

: NETT
  CR ." IHRE FRAGE IST SEHR INTERESSANT"
  CR ." ES WIRD MEINEM KOLLEGEN MEIER"
  CR ." EINE EHRE SEIN, SIE ZU BEANTWORTEN"
;
FIND NETT VARIABLE METHODE
```

So, wie die Dinge jetzt stehen, enthält **METHODE** die Kompilieradresse von **NETT**. Aber wenn Sie mit dem linken Fuß aufgestanden sind oder Ihr Kollege den Fragesteller wieder an Sie zurückverweist, könnten Sie eingeben

```
FIND GROB METHODE !
```

Mit `METHODE @ EXECUTE`

finden Sie dann die passende Antwort.

122

Das ist zwar sehr nützlich, aber bei Verwendung von **REDEFINE** stellt es doch eine erhebliche Beeinträchtigung dar. **REDEFINE** ändert die Kompilieradresse eines Worts, aber es weiß nicht, wie es die Werte von Variablen wie **METHODE** festlegen soll. Das müssen Sie selbst tun. Geändert werden die Kompilieradressen der neueren Wörter.

In ähnlicher Weise kann auch **LOAD** Kompilieradressen verändern. Wenn bereits ein Wörterbuch gespeichert ist und Sie ein neues vom **Band** laden, werden alle Kompilieradressen des neuen Wörterbuches verändert.

KAPITEL 24

SPEICHERAUSLEGUNG

Bestimmte Speicherbereiche haben definierte Aufgaben.
Wir lassen die festgelegte Speichereinteilung folgen:

Adressen Hexadezimal	Dezimal	Inhalt
Ø - 1FFF	Ø - 8191	ROM. Die gespeicherte Information kann nicht überschrieben werden (es sei denn, Sie bringen Ihren ACE zum Röntgen). Computer-Programme incl. eingebaute FORTH-Wörter.
2ØØØ - 23FF	8192 - 9215	RAM. Der gleiche Speicherinhalt wie in den Adressen 24ØØ - 27FF. Jedes dieser Bytes hat also zwei Adressen. Je nach gewählter Adresse hat Lesen oder Schreiben unterschiedliche Wirkung. Die niedrigere Adresse gibt FORTH-Programmen eine Priorität, die höhere Adresse bevorzugt die Schaltkreise für die Bildschirmsteuerung. S.Kap.14,Üb.1
24ØØ - 26FF	9216 - 9983	Video RAM. Er enthält das Fernsehbild mit dem ASCII Code jedes der 24*32 = 768 Zeichen.
27ØØ	9984	**RAM, enthält Byte Ø**
27Ø1 - 27FF	9985 - 1Ø239	RAM, enthält den PAD (s.Kap.16)
28ØØ - 2BFF	1Ø24Ø - 11263	RAM, der gleiche Inhalt wie der Speicherbereich 2CØØ - 2FFF und gleicher Auswirkung wie beim Video RAM.
2CØØ - 2FFF	11264 - 12287	RAM, enthält Zeichensatz, die Punktmuster für 128 Zeichen (siehe Kap.12). In diesen Speicher kann nur geschrieben werden. Lesen ist nicht möglich.
3ØØØ - 3BFF	12288 - 15359	Drei identische Kopien des RAM 3CØØ bis 3FFF.
3CØØ - 3C3F	1536Ø - 15423	RAM, enthält die Systemvariablen. die genauere Beschreibung dieser Variablen folgt unten.
3C4Ø - 3FFF	15424 - 16383	RAM, enthält das Wörterbuch, den DATA- und den RETURN-stack

Das Wörterbuch beginnt bei 3C4Ø mit dem Leitwort **FORTH** und fährt mit Ihren eigenen Wörtern fort. Sein Ende wird lediglich durch die Anzahl und den Umfang der gespeicherten Wörter bestimmt. Auf das Wörterbuch folgen 12 unbenutzte Bytes zum Schutz gegen einen Unterlauf (engl.:underflow) des stack (d.h. der stack ist völlig leer). Danach beginnt der stack aufsteigend.

Der RETURN-stack beginnt bei 3FFF und wächst rückwärts in Richtung des DATA-stack. Wenn beide zusammenkommen, ist kein Speicherplatz mehr vorhanden und der Computer meldet ERROR 1.

```
-------------------------------------------------------------
| Wörterbuch | 12 Bytes | stack -----> <----- RETURN-stack|
-------------------------------------------------------------
3C4Ø                                                    3CFF
```

Mit einer Speichererweiterung (an der Rückfront aufsteckbar) haben Sie wesentlich mehr Platz für Programme und stacks zur Verfügung. Er beginnt mit Adresse 4ØØØ (16384) und geht bis FFFF (65535).

Systemvariablen

In der nachfolgenden Liste sind neben den Hex- und Dezimaladressen noch englischsprachige Abkürzungen angegeben. Diese Namen haben (außer wenigen **fett gedruckten**, wie z.B. **BASE**), für den ACE keine Bedeutung, sie sollen lediglich als Gedächtnisstütze dienen.

Adresse Hex	Dezimal	Name	Länge	Bedeutung
3CØØ	15364Ø	FP-WS	19	Arbeitsbereich für Gleitkommaarithmet.
3C13	15379	LISTWS	5	Arbeitsbereich für **LIST** und **EDIT**
3C18	15384	RAMTOP	2	Die höchste Adresse im RAM. Wenn Sie am Ende Platz reservieren wollen,der für Wörterbuch und stacks nicht verfügbar sein soll, können Sie dessen Startadresse in RAMTOP einsetzen und QUIT eingeben. Mit QUIT wird der RETURN-st ack gelöscht und bei der Adresse aus RAMTOP neu begonnen.
3C1A	15386	HLD	2	Adresse des letzten Bytes für eine formatierte Ausgabe im PAD (benutzt durch Wörter wie **#**, **HOLD** etc.)
3C1C	15388	SCRPOS	2	Cursor-(Anzeige-) Position. Die Adresse, an der das nächste Zeichen angezeigt werden soll (**TAB** in Kap.12 benutzt diese Variable).
3C1E	15399Ø	INSCRN	2	Startadresse der aktuellen logischen Zeile im Eingabebereich
3C2Ø	15392	CURSOR	2	Adresse des CURSORS im Eingabebereich
3C22	15394	ENDBUF	2	Endadresse der aktuellen logischen Zeile im Eingabebereich

Adresse Hex	Dezimal	Name	Länge	Bedeutung
3C24	15396	L_HALF	2	Anfangsadresse des Eingabebereichs. Der Bereich selbst ist im Video-RAM gespeichert.
3C26	15398	KEYCOD	1	ASCII Code der zuletzt gedrückten Taste.
3C27	15399	KEYCNT	1	Beide Systemvariablen werden von der Programmroutine benutzt die das
3C28	154ØØ	STATIN	1	Tastenfeld abfragt.
3C29	154Ø1	EXWRCH	2	Normalerweise Ø. Kann aber geändert werden um auszugebende Daten an ein bestimmtes Gerät zu addressieren (z.B. Drucker). Vergleichen Sie das folgende Kapitel.
3C2B	154Ø3	FRAMES	4	Eine doppeltlange Zahl, die die Zeit vom Einschalten des ACE in 1/5Ø-Sek. zählt. Kann als Uhr benutzt werden. In Übung 1 haben wir Wörter für das Stellen und Anzeigen der Uhrzeit angegeben

Vorsicht!
Bei Überlauf der Uhr über Hex FFFF FFFF wird die nächste Systemvariable XCOORD in Mitleidenschaft gezogen. BEEP und Bandoperationen unterbrechen den Zähler zeitweise.

3C2F	154Ø7	XCOORD	1	Die von PLOT zuletzt verwendete X-Koordinate (s.Kapitel 13, Übung 1, DRAW.
3C3Ø	154Ø8	YCOORD	1	Die von PLOT zuletzt verwendete Y-Koordinate.
3C31	154Ø9	CURRENT	2	Parameteradresse des Leitworts der gerade verwendeten Wörterbuchs (s.Kap. 22).
3C33	15411	CONTEXT	2	Parameter-Adresse des Leitworts des Context-Wörterbuchs (s.Kap.22).
3C35	15413	VOCLNK	2	Adresse des 4.Bytes im Parameterfeld (Wortverkettung=Vocabulary linkage) im Leitwort des zuletzt definierten Wörterbuchs (s.Kapitel 22).
3C37	15415	STKHOT	2	Adresse des Bytes, bei dem der nächste Eintrag ins Wörterbuch beginnt, d.h. Ende des aktuellen Wörterbuchs +1. 15415 @ kann durch HERE ausgeführt werden.
3C39	15417	DICT	2	Die Adresse des Längenfeldes im neuesten Wort des Wörterbuchs. Wenn das Längenfeld richtig geladen ist, kann DICT =Ø sein.

Adresse Hex	Adresse Dezimal	Name	Länge	Bedeutung
3C3B	15419	SPARE	2	Adresse des ersten Bytes nach dem TOS. Mit 15419 @ erhalten Sie die Adresse des obersten Eintrags. In Übung 2 bringen wir ein Wort, das den gesamten Inhalt des stack anzeigt ohne ihn zu zerstören.
3C3D	15421	ERR_NO	1	Normalerweise 255, d.h. "kein Fehler". Wenn ABORT verwendet wird kann ERR_NO einen Wert zwischen Ø und 127 annehmen. Dann wird angezeigt ERROR, gefolgt von ERR_NO
3C3E	15422	FLAGS	1	Zeigt den Zustand verschiedener Systemteile an, wobei jedes Bit eine bestimmte Zuordnung hat. Ist Bit 2 auf 1 gesetzt, zeigt es eine unvollständige Definition am Ende des Wörterbuchs an. Mit ABORT wird die Definition entfernt Bit 3 auf 1 zeigt an, daß Ausgabedaten in den Eingabebereich übertragen werden sollen. Bit 4 auf 1 markiert INVISIBLE-Modus. Bit 6 auf 1 markiert Kompiler-Modus.
3C3F	15423	BASE	1	Zahlenbasis

Übungen:

1. Wörter, mit denen Sie die Uhrzeit stellen und wieder ablesen können

```
: STELLEN
  ( Stunden, Minuten - )
  BEGIN
    INKEY Ø= ( Warten auf ENTER)
  UNTIL
  CR ." Bitte Taste betätigen"
  SWAP 6Ø * + ( Zeit in Minuten)
  3ØØØ U* ( Zeit in fünfzigstel Sek.,Doppellänge)
  BEGIN
    ( Warten auf Tastendruck)
    INKEY
  UNTIL
  Ø 154Ø3 !
  154Ø5 ! 154Ø3 !
;
```

(In Übung 5 erfahren Sie den Trick, mit dem wir hier arbeiten müssen)

```
: MINSEC
  ( Sekunden oder Minuten doppelt lang - Minuten oder
Stunden einfach lang)
  ( Schreibt Sekunden oder Minuten in den PAD)
  6Ø U/MOD SWAP
  Ø # # DROP DROP ( Sekunden oder Minuten)
  ASCII : HOLD
;
```

Fortsetzung Folgeseite

```
: ZEIT
  ( zeigt mit formatierter Ausgabe an)
  154Ø3 @ 154Ø5 @
  OVER OVER D+ ( hundertstel Sekunden)
  <# # # ( Sekundenbruchteile)
  ASCII . HOLD ( Im stack jetzt Zeit in Sekunden)
  MINSEC ( Sek.anzeigen,Zeit in Minuten im stack lassen)
  Ø MINSEC ( Minuten anzeigen, Stunden des stack)
  Ø #S #> TYPE
;
```

2. Anzeige im stack

```
: .S
  15419 @ HERE 12 +
  ( Anfang, Ende)
  OVER OVER -
  IF ( stack nicht leer)
    DO
     I @ . 2
    +LOOP
  ELSE
    DROP DROP
  THEN
;
```

3. Geben Sie ein

```
: SYSVAR
  ( zeigt ständig alle Systemvariablen an)
  CLS
  BEGIN
   Ø Ø AT 1536Ø 8Ø TYPE
  Ø
  UNTIL
;
```

Sie sehen sofort, daß FRAMES ständig zählt. Die zweite sich ändernde Variable ist KEYCNT. Links davon steht KEYCOD. Sie zeigt Daten an, wenn Sie eine Taste betätigen. Am Ende erkennen Sie den Header von FORTH.

Warum flackert die Folgezeile alle 5 Sekunden? (Hinweis: Was geschieht, wenn das zweite Byte von FRAMES den Wert 13 annimmt, den ASCII-Code für Zeilenvorschub?)

4. Der Zähler FRAMES, der die Wechsel des Fernfehbildes zählt, wird nicht genau jede 1/5Ø Sekunde weitergeschaltet, sondern jede 624/625 fünfzigstel Sekunde. Das Wort **TIME** gewinnt dadurch jedesmal in 625 Sekunden eine Sekunde, pro Tag etwa 2 1/2 Minuten. Für eine korrekte Zeitangabe muß man dies korrigieren.

5. Während Sie FRAMES mit

```
154Ø3 @ 154Ø5 @
```

lesen, könnte das kleinerwertige Byte gerade den Wert 65535 (d.h. sein Maximum) haben. Während des Lesevorgangs wird FRAMES weitergezählt: Die Bytes ab 154Ø3 werden zu Ø, und die Bytes ab 154Ø5 um 1 erhöht. Ihre Zahl ist infolgedessen um 65535 zu hoch (etwa 2Ø Minuten). Das kann Ihnen sogar während der Ausführung von @ passieren. Sie können diesen bösen Fehler umgehen, indem Sie FRAMES zweimal lesen und den <u>kleineren</u> der beiden Werte nehmen.

128

Etwas ähnliches kann auch in **STELLEN** geschehen. Wir setzen zunächst die zwei Bytes bei 154Ø3 auf Ø, so daß wir keine Überschneidung mit den Werten erhalten, die wir dann eingeben.

5. Im Kapitel 2Ø haben wir festgestellt, daß bei Verwendung von **DEFINER** ein Fehler während der Definition eines anderen Worts dieses halbdefinierte Wort im Speicher stehen läßt. Deshalb brauchten wir auch in Kapitel 21 das Wort **NACHRICHT**, falls ein Fehler während der Bereichsdefinition aufgetreten wäre.

   Bit 2 in FLAGS erzwingt die Löschung der teilweisen Definition, wenn **ABORT** ausgeführt wird. Das Wort NACHRICHT hat folgende Form

```
: NACHRICHT
  ( setzt Bit 2 von FLAGS auf 1 und bricht ab)
  15422 C@ 4 OR 15422 C! ABORT
;
```

129

KAPITEL 25

MASCHINENCODE

Man könnte nach allem Vorangegangenen meinen, daß der ACE ein Computer ist, der FORTH versteht. In Wahrheit hat er keine Ahnung, was FORTH ist. Sein Herz ist ein Mikroprozessor, ein Schwerarbeiter, der alles tut, was es zu tun gibt: Er fragt die Tastatur ab, übersetzt die Eingabe, führt sie aus und gibt die Ergebnisse wieder aus.

Der Mikroprozessor wird auch CPU (engl.: Central Processing Unit= Zentraleinheit) genannt. Seine Arbeitsanweisungen entnimmt er dem ROM.

Diese Anweisungen können (natürlich) nicht in FORTH geschrieben werden. Sie sind vielmehr in der Maschinensprache festgelegt, der einzigen Sprache, die der Prozessor wirklich versteht. Jeder Befehl ist in Bytes verschlüsselt. Beim ersten Einschalten muß der Computer zunächst ganz am Anfang beginnen. Er holt sich ein Byte aus Adresse Ø, führt den darin verschlüsselten Befehl aus, kommt danach zum nächsten und wird so vom festgelegten Ablauf geführt. In den verschlüsselten Befehlen ist auch definiert, wie FORTH-Wörter zu interpretieren und auszuführen sind.

Wollten wir uns mit dem Z8Ø-Maschinencode ausführlich befassen, würden wir ein weiteres Buch dieses Umfangs füllen. Darum empfehlen wir, bei Interesse sich ein Buch zu diesem Thema zu beschaffen. Wenn Sie sich aber mit den folgenden Ausführungen begnügen wollen, dann dürfen Sie nicht erwarten, alles zu verstehen.

Das Programmieren in Maschinensprache hat vor allen anderen Sprachen drei Vorteile: Die Programme sind schneller, brauchen weniger Platz, und sie kommen in die verstecktesten Ecken des Computers. FORTH ist in allen drei Bereichen schon sehr gut, so daß Sie nicht unbedingt etwas anderes brauchen. Aber manchmal kann die Maschinensprache doch recht nützlich sein.

Um die Maschinensprache einzusetzen, brauchen Sie das FORTH-Wort CALL (Adresse –).

CALL nimmt die Adresse eines Maschinencodes vom stack und führt den dort verschlüsselten Befehl aus. Dies dauert solange, bis es einen Befehl jp (iy) findet. (Jeder Befehl wird in zwei Formen dargestellt: einmal in Maschinencode, zum anderen auch lesbar in einer mnemotechnischen Form, also einer Gedächtnisstütze. jp (iy) ist diese mnemotechnische Form). Der Maschinencode für jp (iy) besteht aus zwei Bytes, 253 und 233 (s.Anhang A).

Am einfachsten läßt sich der Maschinencode im Parameterfeld eines FORTH-Worts unterbringen. Dafür benutzen wir ein Definitionswort

```
DEFINER CODE
DOES>
  CALL
;
```

Die Aufgabe eines Worts, das mit CODE definiert wird, ist es, den Maschinencode in seinem Parameterfeld ablaufen zu lassen. Das Feld wird mit C, erzeugt.

Wir geben ein Beispiel: Der Maschinenbefehl halt (Code 118) hält den Prozessor an, bis er ein Signal interrupt erhält. Im ACE gibt der Computer dem Prozessor jede fünfzigstel Sekunde einen Interrupt. halt kann für terminierte Unterbrechungen benutzt werden.

Sie brauchen dazu den Maschinencode

```
halt        118
jp (iy)     253,233
```

Bringen wir also diese drei Bytes in das Parameterfeld eines Wortes HALT.

```
CODE HALT 118 C, 253 C, 233 C,
```

Wenn Sie jetzt **HALT** ausführen, ruft sein Aktionsteil diesen Maschinencode mit **CALL** auf: Jetzt können wir **PAUSE** definieren

```
: PAUSE
  ( Länge der Pause in 1/50 Sekunden - )
  0
  DO
    HALT
  LOOP
;
```

Wenn Sie schon etwas über die Z80-Maschinensprache wissen, werden Sie es vielleicht für nützlich halten, mehr über die Restart-Befehle zu erfahren:

rst 8 (Code 207) gibt ein Zeichen aus dem Register A aus. Normalerweise geht dies auf den Bildschirm. Sie können dies aber dazu benutzen, ein externes Gerät anzusprechen, in dem Sie die Systemvariable EXWRCH aktivieren (s.Kapitel 24). Dieser Restart benutzt das Register A und die Hilfsregister B,C,D,E,H und L.

rst 16 (Code 215) legt das Registerpaar DE auf den stack und verwendet die Register H und L.
Im Gegensatz zu anderen FORTH-Systemen im Z80-Prozessor verwendet ACE den Maschinen-stack als RETURN-stack und baut sich den DATA-stack gesondert auf.

rst 24 (Code 233) überträgt den TOS in die Register DE und verwendet dazu Register H und L.

rst 32 (Code 231) ist im Wesentlichen **ABORT**. Auf den Restartbefehl sollte ein Byte mit Fehlercode folgen; dieses wird in die Systemvariable ERR_NO angelegt, bevor **ABORT** ausgeführt wird.

Einige Warnungen an Fachleute:

1. Die Register IX und IY dürfen benutzt werden. Ihr ursprünglicher Inhalt muß aber am Ende wieder zurückgeschrieben werden (beachten Sie später die Bemerkungen zu IY). Alle übrigen Register stehen zur freien Verfügung, aber machen Sie keinen Unsinn mit dem stack-Pointer SP.

2. **REDEFINE** und **LOAD** können Wörter im Wörterbuch transportieren (s.Kapitel 23, Übung 1). Wenn Sie Maschinenbefehle im Wörterbuch halten, wie oben mit **HALT** gezeigt, müssen diese entweder verschiebbar (relocatable) sein, oder Sie müssen sehr vorsichtig mit **REDEFINE** und **LOAD** umgehen.

Die Tatsache, daß ein Programm in Maschinencode, das mit **CALL** aufgerufen wird, mit jp (iy) endet, trifft auch für alle in Maschinencode definierten FORTH-Wörter zu. Solche Wörter heißen Stammwörter. Es gibt viele Stammwörter unter den 142 FORTH-Wörtern im ROM. Durch Veränderung des Registers IY können Sie das Ende jedes Stammworts beeinflussen.

Im Normalfall weist das Register IY auf bestimmte Maschinencodes hin, die Fehler prüfen, bevor die Arbeit fortgesetzt wird. Geprüft werden: Space (Abstand zwischen DATA- und RETURN-stack), stack underflow (Datenunterlauf im stack), und die BREAK-Taste. Sie können Zeit gewinnen, indem Sie IY immer auf das nächste auszuführende Wort hinweisen lassen. Zwei Wörter, **FAST** und **SLOW** helfen Ihnen dabei.

FAST (-) schaltet alle Fehlerprüfungen aus. Dadurch werden die Programme um ca. 25 % schneller.

SLOW (-) schaltet die Fehlerprüfungen wieder ein.

Denken Sie aber immer daran: **FAST** ist gefährlich. Sie sollten es nur im Zusammenhang mit sorgfältig getesteten Programmen benutzen. Insbesondere wird natürlich auch BREAK nicht mehr abgefragt.

Zusammenfassung:

Z80-Maschinencode

FORTH-Wörter **CALL**, **FAST**, **SLOW**

Übungen:

1. Arbeiten Sie **CODE** weiter aus, damit Sie mit seiner Hilfe das Maschinencode-Parameterfeld einfacher aufbauen können. Sie könnten z.B. die Bytes des Maschinencodes in den stack stellen und ihm sagen, wieviele vorhanden sind. Vielleicht lassen Sie jp (iy) automatisch einsetzen. Im FORTH wäre es kein Problem, ein komplettes Wörterbuch **ASSEMBLER** anzulegen, das Ihnen erlaubt, die Abkürzungen der Maschinencodes statt der Zahlenkombinationen zu verwenden. Im **ASSEMBLER** ist dann z.B. **HALT** definiert als

```
: HALT
  118 C,
;
```

   Mit **CODE** würde **ASSEMBLER** zum Context-Wörterbuch.

2. Ändern Sie **PAUSE** so, daß die Pause bei Tastendruck beendet wird.

3. Wenn Sie wissen wollen, wie das Codefeld eines Worts funktioniert, können wir es jetzt erklären. Es ist die Adresse einiger Befehle in Maschinencode, die bei Ausführung des Worts ablaufen. Sie bilden das komplette Wort (bei Stammwörtern), oder sie legen fest, wie eine Folge von kompilierten FORTH-Wörtern ablaufen soll (bei Doppelpunkt-Definitionen).

   In einem Stammwort ist das Codefeld die Adresse des Parameterfeldes und das Parameterfeld enthält den Maschinencode. **DUP** ist ein Stammwort, deshalb führen

```
FIND DUP 2+ CALL
```

   und

```
FIND DUP @ CALL
```

beide **DUP** aus. Natürlich geht das auch einfacher.

KAPITEL 26

WIR BAUEN DEN ACE WEITER AUS

Über die Steckerleisten an der Rückseite kann zusätzlich Elektronik (Peripherie) an den ACE angeschlossen werden. Normalerweise verwendet man dazu fertige Geräte, wie z.B. die von uns lieferbaren Zusatzspeicher oder einen Drucker. Es gibt auch Programme, mit denen solche Peripheriegeräte gesteuert werden können.

Sie können aber auch selbst Peripheriegeräte aller Art bauen und anschließen. Dazu müssen Sie nur wissen, wie Ihr Gerät mit Ihrem Programmen zusammenarbeitet und wie es physisch angeschlossen werden muß.

Zunächst zur Kommunikation. Der Z8Ø-Prozessor unterhält sich mit dem Rest der Welt, indem er Spannungen über Leitungen anbietet. Grob gesagt, gibt es zwei Spannungszustände, hoch oder niedrig, die als Ø oder 1 interpretiert werden können. Acht Leitungen in einer Gruppe stellen 1 Byte dar. Es gibt eine solche Gruppe, die wir den Datenbus nennen. Sechzehn weitere Leitungen können eine Ganzzahl bilden, wir nennen diese Gruppe den Adreßbus.

Der Prozessor liest und schreibt damit Speicherinhalte. Soll ein Byte in den Speicher geschrieben werden, stellt er die Adresse in den Adreßbus und die Information in den Datenbus und gibt ein Signal aus, das den Speicher veranlaßt, die Daten in die angegebene Adresse zu übernehmen. Entsprechend verfährt er auch beim Lesen eines Bytes.

Ganz ähnlich verfährt er auch bei der Kommunikation mit Peripheriegeräten. Es gibt 65536 I/O-Adressen (für Input/Output), die Ports genannt werden. Adressierung, Bereitstellen eines Bytes sowie das Signal an das Peripheriegerät werden genauso durchgeführt wie beim Speicher. Die Peripherie muß diese Signale überwachen.

Für den Kontakt zu den I/O-Ports stehen zwei Wörter zur Verfügung: IN und OUT. Sie sind gleichbedeutend mit den Speicherwörtern C@ und C!.

IN (Adresse - Byte) liest ein Byte von dem Port mit der gegebenen Adresse.

OUT (Byte, Adresse -) schreibt ein Byte zu dem Port mit der gegebenen Adresse.

Vermeiden Sie bei der Festlegung von Port-Adressen für Ihre Geräte die bereits vom ACE benutzten Adressen. Der Computer verwendet die geraden Zahlen, die ungeraden stehen zu Ihrer Verfügung.

Wir haben zwei einfache Anwendungen in einem Paket zusammengefaßt. Die eine steuert eine Verkehrsampel, die andere schaltet ein Relais ein und aus. Sie brauchen nur einen I/O-Port (Adresse 1) und erwarten vom Datenbus die folgende Codierung

1	Rotes Licht einschalten	)
2	Gelbes Licht einschalten	)
3	Rot und gelb einschalten	)
4	Grün einschalten	) und alle anderen Signale aus
8	Relais einschalten	)
Ø	Alles ausschalten	

Dazu brauchen wir folgende FORTH-Wörter

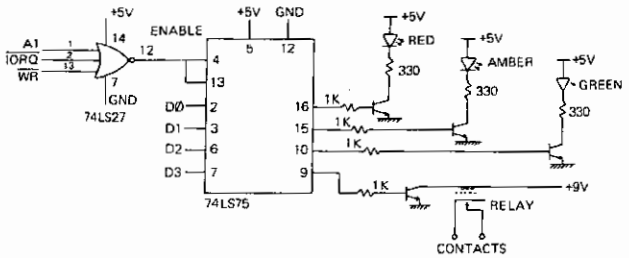
```
: CHANGE
  ( Daten -)
  ( schreibt an I/O Port 1)
  1 OUT
;

: ROFF
  ( - )
  ( schaltet Relais aus)
  Ø CHANGE
;

: RON
  ( -)
  ( schaltet Relais ein)
  8 CHANGE
;

: WAIT
  ( Verzögerung - Verzögerung)
  DUP Ø
  DO
  LOOP
;

DEFINER LICHT
  ( Lampencode -)
  C,
DOES>
  ( Adresse des Lampencodes - )
  C@ CHANGE WAIT
;

1 LICHT ROT
2 LICHT GELB
3 LICHT RT&GB
4 LICHT GRUEN

: FOLGE
  ( Verzögerung - Verzögerung)
  ROT RT&GB GRUEN GELB
;

: ABLAUF
  ( Verzögerung - Verzögerung)
  BEGIN
    FOLGE Ø
  UNTIL
;
```

1ØØØØ **ABLAUF** durchläuft ständig die Ampelschaltung und läßt jede Farbe während der Warteschleife von 1ØØØØ LOOPs eingeschaltet.

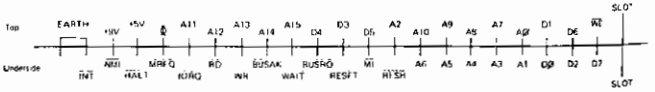
Dazu zeigen wir Ihnen einen Schaltplan für die Elektronik:

134

Traffic Light Controller Use 2N3904 transistors



The signals to this circuitry (A1, IORQ and so on) need to be connected to the computer through an *edge connector*, a connector that clips over the back edge of the circuit board. Here is the arrangement of the signals on the computer.



Wir lassen noch ein Beispiel für Input folgen. Es tastet 6 Schalter ab, von denen jeder ein- (1) oder ausgeschaltet (Ø) sein kann. Das gelesene Byte bringt die Information über die jeweilige Schalterstellung in 6 verschiedenen Bits

```
DEFINER SCHALTER
  ( Bitwert -)
  C,
DOES>
  ( Adresse des Bitwerts - MERKER)
  ( hinterläßt 1 für geschlossenen, Ø für offenen Schalter)
  C@ 1 IN AND Ø= Ø=

 1 SCHALTER SØ
 2 SCHALTER S1
 4 SCHALTER S2
 8 SCHALTER S3
16 SCHALTER S4
32 SCHALTER S5
```

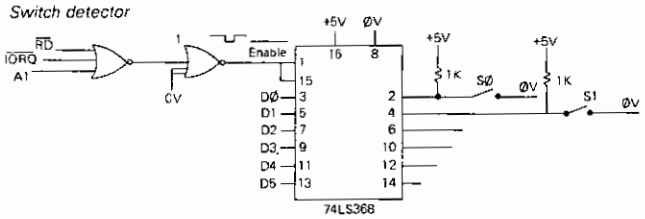
Die Wörter SØ bis S5 prüfen die entsprechenden Schalter und setzen einen MERKER im stack, 1 für den geschlossenen Schalter. Schalter Ø kann als Morsetaste benutzt werden.

```
: MORSE
  BEGIN
    SØ
    IF
      1ØØ 1Ø BEEP
    THEN
    Ø
  UNTIL
;
```

Hier ist ein Wort **CHECK**, das prüft, ob alle Schalter geschlossen sind, und ein anderes, **ALARM**, das ein Signal erzeugt, wenn dies nicht der Fall ist

```
: CHECK
  ( - MERKER)
  1 IN 63 AND 63 =
;

: ALARM
  BEGIN
    CHECK Ø=
    IF
      1ØØ 1ØØØ BEEP
    THEN
    Ø
  UNTIL
;
```

Wir haben wieder I/O Adresse 1 verwendet. Dazu der Schaltkreis



Die Schalter S2 bis S5 sind genau wie die Schalter SØ und S1 mit einem 1 k-Widerstand ausgelegt und mit den Stiften 6, 1Ø, 12 und 14 verbunden.

Unsere Peripherie hat einen I/O Port mit der Nummer 1 benutzt. Peripheriegeräte können aber auch Speicheradressen verwenden, sie heißen dann memory-mapped (speichergebundene) Peripheriegeräte. Es ist darauf zu achten, daß sie nicht Speicherplatz benutzen, der intern benötigt wird (s.Kapitel 24). Alle geraden I/O Port-Adressen sind für die interne Verwendung des ACE reserviert, auch wenn er nur acht davon benutzt, nämlich (in Hex) FEFE, FDFE, FBEE, F7FE, EFFE, DFFE, BFFE und 7FFE. Das niedrigerwertige Byte ist immer FE. Das höherwertige Byte hat immer ein Bit Ø und alle anderen 1.

Wird von einem dieser Ports gelesen, dann wird

a) eine halbe Reihe der Tastatur abgefragt. Welche Hälfte dies ist, bestimmt die Ø im höherwertigen Teil der Port-Adresse. Die niedrigerwertigen 5 Bits zeigen, ob eine Taste gedrückt ist (Ø) oder nicht (1).

b) Das Signal am Stecker EAR wird in Bit 5 gelesen.

c) Die Lautsprechermembran wird eingeschaltet.

Wird an einem dieser Ports geschrieben dann
- wird Bit 3 der Daten an den Stecker MIC übergeben,
- der Lautsprecher ausgeschaltet.

 136

Zusammenfassung:

I/O Ports, Port-Adressen

FORTH-Wörter **IN**, **OUT**

Übungen:

1. Überarbeiten Sie das Wort **CHANGE**, so daß es anstatt externe Schaltungen zu betreiben, etwas auf dem Bildschirm anzeigt.

2. Geben Sie ein

```
: KRACH
  [ 16 BASE C! ]
  BEGIN
    FEFE IN FEFE OUT Ø
  UNTIL
;
```

   Schrecklich!

   Warum ändert sich der Ton, wenn Sie SHIFT drücken? (Denken Sie daran, daß der Computer ständig auf BREAK prüft).

3. Verwenden Sie das Relais im Verkehrsampel-Spiel, um Lichter zu schalten. Noch besser, wenn Ihr Kassettenrecorder eine Buchse hat, über die ein Mikrofon ein- und ausgeschaltet werden kann, dann versuchen Sie dies mit dem Relais zu tun.

4. Die Signale am Ausgang der Schaltplatine sind weitgehend für diese Art der Z8Ø Computer standardisiert, sie können aber von anderen Typen abweichen. Viele Zusätze zum Sinclair ZX81 arbeiten auch mit dem ACE, wenn Sie die entsprechenden Anschlüsse miteinander verdrahten. Für diesen Fall ist es am zweckmäßigsten, einen Adapter mit einer Steckerleiste für den ACE und einer gedruckten Schaltung für den Anschluß des Zusatzgerätes zu bauen und dazwischen die Verdrahtung vorzunehmen.

The RAM C.S. and ROM C.S. lines on the ZX81, and the WE on the Ace, don't need connecting.

137

ANHANG A

DER ZEICHENSATZ

Code	Zeichen	Hex	Z8Ø Assembler	-nach CB	-nach ED
Ø	neue Zeile	ØØ	nop	rlc b	
1	)	Ø1	ld bc,NN	rlc c	
2	)	Ø2	ld(bc),a	rlc d	
3	)	Ø3	inc bc	rlc e	
4	)	Ø4	inc b	rlc h	
5	)	Ø5	dec b	rlc l	
6	)	Ø6	ld b,N	rlc (hl)	
7	) nicht verwendet	Ø7	rlca	rlc a	
8	)	Ø8	ex af,af'	rrc b	
9	)	Ø9	add hl,bc	rrc c	
1Ø	)	ØA	ld a,(bc)	rrc d	
11	)	ØB	dec bc	rrc e	
12	)	ØC	inc c	rrc h	
13	Wagenrücklauf	ØD	dec c	rrc l	
14	)	ØE	ld c,N	rrc (hl)	
15	) nicht verwendet	ØF	rrca	rrc a	
16	▀	1Ø	djnz DIS	rl b	
17	▛	11	ld de,NN	rl c	
18	▜	12	ld (de),a	rld	
19	▄	13	inc de	rl e	
2Ø	▙	14	inc d	rl h	
21	▌	15	dec d	rl l	
22	▚	16	ld d,N	rl (hl)	
23	▗	17	rla	rl a	
24	)	18	jr DIS	rr b	
25	)	19	add hl,de	rr c	
26	)	1A	ld a,(de)	rr d	
27	) nicht verwendet	1B	dec de	rr e	
28	)	1C	inc e	rr h	
29	)	1D	dec e	rr l	
3Ø	)	1E	ld e,N	rr (hl)	
31	)	1F	rra	rr a	
32	SPACE	2Ø	jr nz,DIS	sla b	
33	!	21	ld hl,NN	sla c	
34	"	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	
38	&	26	ld h,N	sla (hl)	
39	'	27	daa	sla a	
4Ø	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	-	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	Ø	3Ø	jr nc,DIS		
49	1	31	ld sp,NN		
5Ø	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	

138

Code	Zeichen	Hex	Z8Ø Assembler	-nach CB	-nach ED
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
6Ø	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	4Ø	ld b,b	bit Ø,b	in b,(c)
65	A	41	ld b,c	bit Ø,c	out (c),b
66	B	42	ld b,d	bit Ø,d	sbc hl,bc
67	C	43	ld b,e	bit Ø,e	ld (NN),bc
68	D	44	ld b,h	bit Ø,h	neg
69	E	45	ld b,l	bit Ø,l	retn
7Ø	F	46	ld b,(hl)	bit Ø,(hl)	im Ø
71	G	47	ld b,a	bit Ø,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
8Ø	P	5Ø	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
9Ø	Z	5A	ld e,d	bit 3,d	adc hl,de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	\	5C	ld e,h	bit 3,h	
93	]	5D	ld e,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	_	5F	ld e,a	bit 3,a	ld a,r
96	£	6Ø	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
1ØØ	d	64	ld h,h	bit 4,h	
1Ø1	e	65	ld h,l	bit 4,l	
1Ø2	f	66	ld h,(hl)	bit 4,(hl)	
1Ø3	g	67	ld h,a	bit 4,a	rrd
1Ø4	h	68	ld l,b	bit 5,b	in l,(c)
1Ø5	i	69	ld l,c	bit 5,c	out (c),l
1Ø6	j	6A	ld l,d	bit 5,d	adc hl,hl
1Ø7	k	6B	ld l,e	bit 5,e	ld hl,(NN)
1Ø8	l	6C	ld l,h	bit 5,h	
1Ø9	m	6D	ld l,l	bit 5,l	
11Ø	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	7Ø	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld (hl),l	bit 6,l	

139

Code	Zeichen	Hex	Z8Ø Assembler	-nach CB	-nach ED
118	v	76	halt	bit 6,(hl)	
119	w	77	ld (hl),a	bit 6,a	
12Ø	x	78	ld a,b	bit 7,b	in a,(c)
121	y	79	ld a,c	bit 7,c	out (c),a
122	z	7A	ld a,d	bit 7,d	adc hl,sp
123	{	7B	ld a,e	bit 7,e	ld sp,(nn)
124	|	7C	ld a,h	bit 7,h	
125	}	7D	ld a,l	bit 7,l	
126	~	7E	ld a,(hl)	bit 7,(hl)	
127	©	7F	ld a,a	bit 7,a	
128	)	8Ø	add a,b	res Ø,b	
129	)	81	add a,c	res Ø,c	
13Ø	)	82	add a,d	res Ø,d	
131	)	83	add a,e	res Ø,e	
132	)	84	add a,h	res Ø,h	
133	)	85	add a,l	res Ø,l	
134	)	86	add a,(hl)	res Ø,(hl)	
135	) nicht verwendet	87	add a,a	res Ø,a	
136	)	88	adc a,b	res 1,b	
137	)	89	adc a,c	res 1,c	
138	)	8A	adc a,d	res 1,d	
139	)	8B	adc a,e	res 1,e	
14Ø	)	8C	adc a,h	res 1,h	
141	)	8D	adc a,l	res 1,l	
142	)	8E	adc a,(hl)	res 1,(hl)	
143	)	8F	adc a,a	res 1,a	
144	□	9Ø	sub b	res 2,b	
145	◪	91	sub c	res 2,c	
146	◪	92	sub d	res 2,d	
147	⬒	93	sub e	res 2,e	
148	◪	94	sub h	res 2,h	
149	◨	95	sub l	res 2,l	
15Ø	▚	96	sub (hl)	res 2,(hl)	
151	▚	97	sub a	res 2,a	
152	)	98	sbc a,b	res 3,b	
153	)	99	sbe a,c	res 3,c	
154	)	9A	sbc a,d	res 3,d	
155	) nicht verwendet	9B	sbc a,e	res 3,e	
156	)	9C	sbc a,h	res 3,h	
157	)	9D	sbc a,l	res 3,l	
158	)	9E	sbc a,(hl)	res 3,(hl)	
159	)	9F	sbc a,a	res 3,a	
16Ø		AØ	and b	res 4,b	ldi
161	inverse !	A1	and c	res 4,c	cpi
162	inverse "	A2	and d	res 4,d	ini
163	inverse #	A3	and e	res 4,e	outi
164	inverse $	A4	and h	res 4,h	
165	inverse %	A5	and l	res 4,l	
166	inverse &	A6	and (hl)	res 4,(hl)	
167	inverse '	A7	and a	res 4,a	
168	inverse (	A8	xor b	res 5,b	ldd
169	inverse)	A9	xor c	res 5,c	cpd
17Ø	inverse *	AA	xor d	res 5,d	ind
171	inverse +	AB	xor e	res 5,e	outd
172	inverse ,	AC	xor h	res 5,h	
173	inverse -	AD	xor l	res 5,l	
174	inverse .	AE	xor (hl)	res 5,(hl)	
175	inverse /	AF	xor a	res 5,a	
176	inverse Ø	BØ	or b	res 6,b	ldir

Code	Zeichen	Hex	Z8Ø Assembler	-nach CB	-nach ED
177	inverse 1	B1	or c	res 6,c	cpir
178	inverse 2	B2	or d	res 6,d	inir
179	inverse 3	B3	or e	res 6,e	otir
18Ø	inverse 4	B4	or h	res 6,h	
181	inverse 5	B5	or l	res 6,l	
182	inverse 6	B6	or (hl)	res 6,(hl)	
183	inverse 7	B7	or a	res 6,a	
184	inverse 8	B8	cp b	res 7,b	lddr
185	inverse 9	B9	cp c	res 7,c	cpdr
186	inverse :	BA	cp d	res 7,d	indr
187	inverse ;	BB	cp e	res 7,e	otdr
188	inverse <	BC	cp h	res 7,h	
189	inverse =	BD	cp l	res 7,l	
19Ø	inverse >	BE	cp (hl)	res 7,(hl)	
191	inverse ?	BF	cp a	res 7,a	
192	inverse @	CØ	ret nz	set Ø,b	
193	inverse A	C1	pop bc	set Ø,c	
194	inverse B	C2	jp nz,NN	set Ø,d	
195	inverse C	C3	jp NN	set Ø,e	
196	inverse D	C4	call nz,NN	set Ø,h	
197	inverse E	C5	push bc	set Ø,l	
198	inverse F	C6	add a,N	set Ø,(hl)	
199	inverse G	C7	rst Ø	set Ø,a	
2ØØ	inverse H	C8	ret z	set 1,b	
2Ø1	inverse I	C9	ret	set 1,c	
2Ø2	inverse J	CA	jp z,NN	set 1,d	
2Ø3	inverse K	CB		set 1,e	
2Ø4	inverse L	CC	call z,NN	set 1,h	
2Ø5	inverse M	CD	call NN	set 1,l	
2Ø6	inverse N	CE	adc a,N	set 1,(hl)	
2Ø7	inverse O	CF	rst 8	set 1,a	
2Ø8	inverse P	DØ	ret nc	set 2,b	
2Ø9	inverse Q	D1	pop de	set 2,c	
21Ø	inverse R	D2	jp nc,NN	set 2,d	
211	inverse S	D3	out (N),a	set 2,e	
212	inverse T	D4	call nc,NN	set 2,h	
213	inverse U	D5	push de	set 2,l	
214	inverse V	D6	sub N	set 2,(hl)	
215	inverse W	D7	rst 16	set 2,a	
216	inverse X	D8	ret c	set 3,b	
217	inverse Y	D9	exx	set 3,c	
218	inverse Z	DA	jp c,NN	set 3,d	
219	inverse [	DB	in a,(N)	set 3,e	
22Ø	inverse \	DC	call c,NN	set 3,h	
221	inverse]	DD	stellt Instrukt. mit ix voran	set 3,l	
222	inverse ↑	DE	sbc a,N	set 3,(hl)	
223	inverse _	DF	rst 24	set 3,a	
224	inverse £	EØ	ret po	set 4,b	
225	inverse a	E1	pop hl	set 4,c	
226	inverse b	E2	jp po,NN	set 4,d	
227	inverse c	E3	ex (sp),hl	set 4,e	
228	inverse d	E4	call po,NN	set 4,h	
229	inverse e	E5	push hl	set 4,l	
23Ø	inverse f	E6	and N	set 4,(hl)	
231	inverse g	E7	rst 32	set 4,a	
232	inverse h	E8	ret pe	set 5,b	
233	inverse i	E9	jp (hl)	set 5,c	
234	inverse j	EA	jp pe,NN	set 5,d	
235	inverse k	EB	ex de,hl	set 5,e	

141

Code	Zeichen	Hex	Z8Ø Assembler	-nach CB	-nach ED
236	inverse l	EC	call pe,NN	set 5,h	
237	inverse m	ED		set 5,l	
238	inverse n	EE	xor N	set 5,(hl)	
239	inverse o	EF	rst 4Ø	set 5,a	
24Ø	inverse p	FØ	ret p	set 6,b	
241	inverse q	F1	pop af	set 6,c	
242	inverse r	F2	jp p,NN	set 6,d	
243	inverse s	F3	di	set 6,e	
244	inverse t	F4	call p,NN	set 6,h	
245	inverse u	F5	push af	set 6,l	
246	inverse v	F6	or N	set 6,(hl)	
247	inverse w	F7	rst 48	set 6,a	
248	inverse x	F8	ret m	set 7,b	
249	inverse y	F9	ld sp,hl	set 7,c	
25Ø	inverse z	FA	jp m,NN	set 7,d	
251	inverse {	FB	ei	set 7,e	
252	inverse |	FC	call m,NN	set 7,h	
253	inverse }	FD	stellt Instrukt. mit iy voran	set 7,l	
254	inverse ∿	FE	cp N	set 7,(hl)	
255	inverse ©	FF	rst 56	set 7,a	

142

ANHANG B

FEHLERANZEIGEN

Es gibt zwei Arten von Fehlern. Die harmlosere tritt auf, wenn der Computer im Eingabebereich ein undefiniertes Wort erkennt: er verwandelt die MARKE in ▯ und gibt Ihnen die Chance zur Korrektur. Beachten Sie bitte, daß ▯ nicht immer einen Fehler anzeigt, sondern daß dieses Zeichen auch unter anderen Umständen auftritt, z.B. wenn Sie **EDIT** benutzen und der Computer Ihnen mit ▯ anzeigt, daß Sie im Eingabebereich Änderungen vornehmen können.

Bei der zweiten Art handelt es sich um echte Fehler, und diese werden durch ERROR und dem zugehörigen Fehlercode auf dem Bildschirm angezeigt. Diese Fehleranzeigen mit ERROR.. löschen den DATA-stack, den RETURN-stack und den Eingabebereich und gleichzeitig wird eine unvollständige Definition im Wörterbuch gelöscht.

Hier nun die einzelnen Fehlercodes mit der Erklärung ihrer Bedeutung und evtl. Ursachen.

Fehlercode bedeutet:

1 Nicht genügend Speicherplatz. Entweder haben Sie den DATA-stack oder den RETURN-stack überladen, oder Sie haben versucht, eine Eintragung ins Wörterbuch zu machen, für die dort kein Platz mehr war. Siehe Kapitel 5.

2 "Underflow" im DATA-stack. Der Computer hat weniger als keine Zahl auf dem stack. Siehe Kapitel 5, Übung 4.

3 Sie haben BREAK getastet. BREAK ist normalerweise SHIFTED SPACE; während Sie mit Cassette arbeiten oder während einer **BEEP**-Operation funktioniert die Taste
auch ohne SHIFT als BREAK.

4 Sie haben versucht, außerhalb einer Wortdefinition ein Wort zu benutzen, welches nur innerhalb einer Wortdefinition zugelassen ist (Compiling-Wort). Kann verursacht werden durch IF, **ELSE**, THEN, BEGIN, UNTIL, WHILE, REPEAT, DO, LOOP, +LOOP, ; , **DOES>** , **RUNS>** , (, ." oder durch ein compilierendes Wort, welches durch COMPILER definiert wurde.

5 Ein Wort ist nicht richtig strukturiert. IF...THEN, IF..ELSE..-THEN, BEGIN...UNTIL, BEGIN...WHILE...REPEAT, DO...LOOP, **DO**...-+LOOP, :...;, **DEFINER...DOES> ...**; und **COMPILER...RUNS> ...**; müssen in richtiger Folge und vollständig verwendet werden. Siehe Kapitel 10.

6 Der Name eines neuen Wortes ist entweder zu kurz (d.h. er fehlt völlig), oder er ist zu lang (64 Zeichen oder mehr). Kann auftreten bei :, **DEFINER**, **COMPILER**, **CREATE**, **CONSTANT**, **VARIABLE** oder bei irgendeinem von Ihnen selbst definierten Wort, bei dem Sie **DEFINER** benutzt haben.

7 Sie haben **PICK** oder **ROLL** mit 0 oder negativem Operanden benutzt. Beachten Sie bitte, daß ein Operand -32768 nicht zu ERROR 7 führt. Siehe Kapitel 6, Übung 2.

143

8 Overflow bei Gleitkommaarithmetik: Das Rechenergebnis ist zu groß für den im ACE reservierten Gleitkommabereich. Kann auftreten bei F+, F-, F* oder F/. Tritt immer auf, wenn Sie durch Ø zu dividieren versuchen.

9 Versuch, in den Eingabebereich zu schreiben. Kann verursacht werden bei **AT** oder PLOT.

1Ø Bandfehler.

a) bei **SAVE** oder **BSAVE** - entweder kein Datenname vergeben oder keine Daten zur Sicherung vorhanden.

b) bei **VERIFY** oder **BVERIFY** - Daten sind nicht (richtig) auf auf Band angekommen

c) bei **LOAD** - der Datenbestand ist zu lang für die verbleibende Speichergröße

d) bei **BLOAD** oder **BVERIFY** - der Datenbestand ist zu groß für die vorher spezifizierte Länge.

e) bei **LOAD, VERIFY, BLOAD** oder **BVERIFY** - Lesefehler. Entweder durch (teilweise) zerstörte Daten oder defektes Band.

11 Fehler bei **REDEFINE** oder **FORGET**.
Bei **REDEFINE**: Entweder ist das neueste Wort im aktuellen Vokabular nicht auch das neueste im gesamten Wörterbuch; oder das alte Wort wird nicht gefunden (oder ist im Festspeicher); oder das alte Wort wurde mit **DEFINER** oder **COMPILER** definiert, nicht aber das neue; oder es ist nicht mehr genügend Platz im Speicher (man benötigt soviel Platz im Speicher, daß man zwei Kopien des neuen Wortes unterbringen kann, nachdem das alte Wort entfernt wurde).
Bei **FORGET**: Die im CONTEXT- und im CURRENT-Vokabular gespeicherten Texte sind verschieden.

12 Unvollständige Definition im Wörterbuch - bestimmte Operationen sind dann nicht erlaubt. Kann verursacht sein bei **REDEFINE**, bei einer Bandoperation oder bei irgendeinem definierenden Wort (wie unter ERROR 6 aufgeführt). Dieser Fehler behebt sich selbst, indem er die unvollständige Definition aus dem Wörterbuch entfernt. Wahrscheinlich haben Sie eine Definition gestartet (mit :, **DEFINER** oder **COMPILER**), kamen mit [zum interpret-modus und haben dann die nicht zulässige Operation versucht.

13 Wort nicht gefunden (oder ist im Festspeicher oder ist ein festverdrahtetes FORTH-Wort. Kann auftreten bei FORGET, **LIST** oder **EDIT**.

14 Wort ist nicht auflistbar, tritt auf bei **LIST** oder **EDIT**. Nur Wörter, die mit : , DEFINER oder COMPILER definiert wurden, sind auflistbar.

ANHANG C

DER JUPITER ACE - ZUM NACHSCHLAGEN

Zeichen und Tastatur

Der ACE benutzt den ASCII-Zeichensatz mit folgenden speziellen Vorschriften:

Code	Bedeutung
Ø	Trennt logische Zeilen im Eingabebereich
13	Carriage return (Wagenrücklauf)
16-23	Grafische Zeichenmuster
16	▀
17	▜
18	▛
19	▟
2Ø	▙
21	▌
22	▚
23	▗
96	£
127	©
128-255	Inverse-video-Versionen der Zeichen Ø bis 127

Die Formen der Zeichen Ø bis 127 sind im RAM gespeichert und können vom Benutzer selbst neu definiert werden. Jede Zeichenform besteht aus einem Raster von 8x8 Punkten und ist in 8 hintereinanderfolgenden Bytes gespeichert, mit der obersten Punktereihe in der Hex-Adresse 2CØØ + 8 * ASCII-Code, und der untersten Punktereihe in der Hex- Adresse 2CØØ + 8 * ASCII-Code + 7.
In jeder Reihe ist der am weitesten links stehende Punkt das signifikante Bit (1=weiß).

Alle Zeichen können von der Tastatur erreicht werden. SHIFT wird für Großbuchstaben und Kontrollfunktionen gebraucht, SYMBOL SHIFT für Sonderzeichen. Drei spezielle Eingabemodi können über die Tastatur ein-und ausgeschaltet werden (unabhängig voneinander): CAPS LOCK schaltet von Klein- auf Großbuchstaben um, INVERSE VIDEO wandelt alle Zeichen in Negativ-Schreibung um, und GRAPHICS wandelt Zeichen in solche mit den Codes Ø bis 31, oder 128 bis 159 um, indem es die Bits 5 und 6 des ASCII-Code ersetzt. Alle Zeichen können durch Dauerdruck auf die jeweilige Taste wiederholt werden.

Der Eingabebereich belegt die unterste Zeile auf dem Bildschirm und wird bei größeren Eingaben nach oben erweitert. Alle Eingaben werden sofort links von der MARKE angezeigt. Die MARKE zeigt durch ihre Form auch den jeweiligen Eingabemodus an: Normal als Viereck, C für CAPS LOCK, G für GRAPHICS, und ? wenn der Computer die Aufforderung ausgibt, etwas im Eingabebereich ändern zu können. Die MARKE kann mit den Tasten SHIFTED 5,6,7 und 8 bewegt werden. DELETE löscht das Zeichen unmittelbar links von der MARKE, und DELETE LINE den gesamten Eingabebereich (aber beachten Sie **EDIT**).

Ist ENTER gedrückt, nimmt der FORTH-Übersetzer Wörter aus dem Eingabebereich, kopiert sie oben auf den Bildschirm, und führt sie aus (aber beachten Sie **INVIS**). Dies wird solange fortgeführt, bis entweder der Eingabebereich leer ist oder ein undefiniertes Wort gefunden wird. SPACE mit SHIFT arbeitet normalerweise wie BREAK und erzeugt ERROR 3.

FORTH Wörterbuch

In den Erläuterungen verwenden wir folgende Kurzzeichen:

n Einfachlange Ganzzahl
d Doppeltlange Ganzzahl
u Zahl (oder Darstellung) ohne Vorzeichen
f Gleitkommazahl
I Sofort auszuführendes Wort
C Verwendung des Worts nur im Compiler-Modus

!	(n,Adresse -) Eine einfachlange Zahl n wird in die gegebene Speicheradresse übertragen
#	(ud1 - ud2) Für formatierte Ausgabe. Nimmt eine Ziffer aus der doppeltlangen Zahl ud1 und legt sie im Pad ab. ud2 ist der Quotient, der entsteht, wenn ud1 durch die Zahlenbasis dividiert wird.
#>	(ud - Adresse,n) Schließt formatierte Ausgabe ab und hinterläßt Adresse und Länge (n) der resultierenden Zeichenfolge im stack.
#S	(ud - Ø,Ø) Verwendet # wiederholt (mindestens einmal), bis die doppeltlange Zahl im stack zu Ø geworden ist.
CI (	Anfang eines Kommentars. Abschluß mit).
*	(n1,n2 - n1*n2)
*/	(n1,n2,n3 - (n1*n2)/n3)
*/MOD	n1,n2,n3 - Rest,Quotient aus ((n1*n2)/n3)
+	n1,n2 - n1+n2)
CI +LOOP	(n -) Wird zusammen mit DO verwendet. Addiert n zum Schleifenzähler und springt zurück, solange der Zähler noch kleiner (für n ≥ Ø) oder größer (für n <Ø) als ein gegebens Limit ist.
,	(n -) Übernimmt eine einfachlange Ganzzahl n ins Wörterbuch.
-	(n1,n2 - n1-n2)
.	(n -) Ausgabe auf Bildschirm mit nachfolgender Leerstelle
CI ."	(-) Gibt den nachfolgenden String aus. Begrenzung durch "
/	(n1,n2 - n1/n2) Division einfachlanger Zahlen mit Vorzeichen
/MOD	(n1,n2 - Rest,Quotient aus n1/n2) Der Rest hat das gleiche Vorzeichen wie der Divident n1.

Ø<	(n - MERKER) Der Merker ist 1, wenn n negativ.
Ø=	(n - MERKER) Der Merker ist 1, wenn n = Ø
Ø>	(n - MERKER) Der Merker ist 1, wenn n positiv.
1+	(n - n+1)
1-	(n - n-1)
2+	(n - n+2)
2-	(n - n-2)
:	Beginn einer Doppelpunkt-Definition
CI ;	Beendet Doppelpunkt-, **DEFINER** und **COMPILER**-Definitionen
<	(n1,n2 - MERKER) Merker ist 1, wenn n1 < n2
<#	(-) Beginnt eine formatierte Ausgabe
=	(n1,n2 - MERKER) Merker ist 1, wenn n1 = n2
>	(n1,n2 - MERKER) Merker ist 1, wenn n1 > n2
>R	(n -) Überträgt den TOS in den RETURN-stack. Mit I kann zurückkopiert werden.
?DUP	(n - n,n) wenn n ≠ Ø (n - n) wenn n = Ø
@	(Adresse - n) Hinterläßt im stack eine einfachlange Zahl n aus der gegebenen Adresse.
ABORT	Löscht DATA- und RETURN-stack, löscht alle unvollständigen Wörter aus dem Wörterbuch. Zeigt ERROR und den Inhalt des Bytes 3C3D (hex) an. Leert den Eingabebereich und gibt die Kontrolle an die Tastatur zurück.
ABS	(n - Absolutwert von n)
ALLOT	(n -) Reserviert n Bytes im Wörterbuch, ohne sie mit Daten zu füllen.
AND	(n1,n2 - n1 **AND** n2) Bitweise arbeitende Boolesche Operation
ASCII	Nimmt das nächste Wort aus dem Eingabebereich und gibt den ASCII-Code seines ersten Zeichens aus. Beispiel: **: STERNE Ø DO ASCII * EMIT LOOP ;** (- ASCII-Code) im Interpreter-Modus (-) im Kompiler-Modus

AT (Zeile,Spalte -)
Setzt die MARKE- (Cursor-) Position auf im stack angegebene Zeile und Spalte. Es gibt 23 Zeilen (Ø bis 22) und 32 Spalten (Ø bis 31). Die Spaltenzahl wird Modulo 32 genommen. ERROR 9 wird angezeigt, wenn man versucht, in den Eingabebereich in der untersten Zeile zu schreiben.

BASE (- 15423)
Eine 1 Byte lange Systemvariable, die die Basis des jeweils verwendeten Zahlensystems enthält.

BEEP (m,n -)
Erzeugt einen Ton im Lautsprecher. 8*m = Schwingungslänge in Mikrosekunden, n = Dauer in Millisekunden.

CI **BEGIN** (-)
Verwendet mit **UNTIL** oder **WHILE...REPEAT**

BLOAD Name (m,n -)
Lädt n Bytes eines Byte Files "Name" von einer Kassette und beginnt bei Speicheradresse m. ERROR 1Ø, wenn das File mehr als n Bytes hat.

BSAVE Name (m,n -)
n Bytes ab Adresse m werden in ein Byte File "Name" auf der Kassette gesichert.

BVERIFY Name (m,n -)
Maximal n Bytes des Bytefiles "Name" auf Kassette werden gegen Daten im RAM, beginnend ab m geprüft. Hat das File mehr als n Bytes, wird ERROR 1Ø gemeldet. BLOAD und BVERIFY benutzen die Adresse, von der aus das Bytefile gesichert wurde, wenn m = Ø ist. Für n = Ø spielt die Länge keine Rolle.

C! (n,Adresse -)
Speichert das weniger signifikante Byte von n in die gegebene Adresse.

C, (n -)
Übernimmt das weniger signifikante Byte von n ins Wörterbuch.

C@ (Adresse - Byte)
Holt den Inhalt der gegebenen Adresse in den stack.

CALL (Adresse -)
Führt den in der Adresse angegebenen Maschinencode aus. Der Code wird durch jp (iy) begrenzt. Beispiel in Hex:
DEFINER CODE DOES> CALL ;
CODE EI FB C, FD C, E9 C;
Das Wort EI ermöglicht Unterbrechungen.

CLS (-)
Löscht den Bildschirm und setzt die MARKE in dessen obere linke Ecke.

148

CI **COMPILER**

Zusammen mit **RUNS>** verwendet. Zur Definition neuer Kompiler-Wörter, also Wörtern, die innerhalb von Definitionen sofort Informationen ins Wörterbuch kompilieren (man arbeitet häufig mit **IMMEDIATE**, aber **COMPILER... RUNS>** läßt sich mit EDIT besser verarbeiten.

Das neue Kompilierwort erzeugt eine 2 Bytes lange Adresse, die auf eine Laufzeit-Aktion hinweist, und im Bedarfsfall ein Operandenfeld.
Format:

```
n COMPILER Name
    Kompilier-Routine
  RUNS>
    Aktions-Routine
;
```

n muß auf dem stack sein und stellt die Zahl der Bytes im Operandenfeld dar. Wenn n = -1, dann müssen die ersten beiden Bytes die Gesamtlänge des restlichen Operandenfeldes beschreiben. "Name" ist der Name des neuen Kompilierworts. "Kompilier-Routine" ist der Teil, der das Operandenfeld erzeugt. Achten Sie darauf, daß die Zahl der Bytes mit dem Wert n im stack übereinstimmt. "Aktions-Routine" führt die Aktion während der Laufzeit aus. Sie wird über die Adresse des Operandenfeldes im stack angesprochen.

Beachten Sie bitte: **LIST** und **EDIT** zeigen Operandenfelder nicht an, und **REDEFINE** korrigiert sie nicht.

CONSTANT Name

(n -)
Definiert eine Konstante mit dem gegebenen Namen und Wert n.

CONTEXT

(- 15411)
Eine Systemvariable, die auf das Context-Wörterbuch hinweist.

CONVERT

(ud1,Adresse1 - ud2,Adresse2)
Sammelt Ziffern aus einem Text in eine doppeltlange Ganzzahl ohne Vorzeichen ud1: Für jede Stelle wird der Akkumulator mit der Basis des Zahlensystems multipliziert und die Ziffer (aus ASCII konvertiert) addiert. Der Text beginnt bei Adresse 1 +1. Adresse 2 ist die Adresse des ersten nicht konvertierbaren Zeichens, ud2 ist die Länge des Akkumulators.

CR

(-)
Gibt einen Zeilenvorschub an den Fernsehschirm.

CREATE Name

(-)
Definiert ein neues Wort mit Header und leerem Parameterfeld. Bei Ausführung legt das neue Wort die Adresse seines Parameterfeldes auf dem stack ab.

149

CURRENT
(- 154Ø9)
Eine Systemvariable, die das (aktuelle) Current-Wörterbuch angibt.

D+
(d1,d2 - d1+d2)
Addition der doppeltlangen Ganzzahlen.

D<
(d1,d2 - MERKER)
Merker ist 1, wenn doppeltlange Ganzzahlen d1 < d2

DECIMAL
(-)
Basis des Zahlensystems auf 1Ø setzen.

DEFINER
Zusammen mit **DOES>** zur Definition neuer Definitionswörter. Das sind Wörter, die ihrerseits neue Wörter definieren können.
Format:

```
DEFINER Name
  Definitionsroutine
DOES>
  Aktionsroutine
;
```

"Name" ist der Name des neuen Definitionsworts. Bei der Ausführung erzeugt es den Header eines neuen Worts und benutzt dessen Definitionsroutine zur Festlegung des Parameterfeldes. Wird dieses neue Wort ausgeführt, kommt sein Parameterfeld auf den stack und der Aktionsteil läuft ab.

DEFINITIONS
(-)
Das Context-Wörterbuch wird auch Current-Wörterbuch.

DNEGATE
(d - -d)
Eine doppeltlange Ganzzahl wird negiert.

CI **DO**
(Grenzwert,Anfangswert -)
Beginnt eine **DO** - Schleife. Der Schleifenzähler wird auf den Anfangswert gesetzt. Grenzwert und Schleifenzähler werden im RETURN-stack gespeichert (Vgl. **LOOP** und **+LOOP**)

CI **DOES>**
Vgl. **DEFINER**

DROP
(n -)
Eine Zahl wird aus dem TOS entfernt.

DUP
(n - n,n)
Der TOS wird dupliziert.

EDIT Name
(-)
Listet das Wort "Name" auf dem Bildschirm für die Aufbereitung (Änderung) an. Es werden maximal 18 Zeilen angezeigt. Mit ENTER können die nächsten Zeilen geholt werden. Am Ende wird mit ENTER die neue Version des Worts ins Wörterbuch gestellt.

Die MARKE (Cursor) kann während des Editierens in den Zeilen bewegt werden. Mit DELETE LINE wird eine Zeile gelöscht.

CI **ELSE**
(-)
Wird mit **IF** und **THEN** verwendet.

EMIT (Zeichen -)
Schreibt ein Zeichen vom TOS auf den Bildschirm.

EXECUTE (Kompilieradresse -)
Führt das Wort mit der gegebenen Kompilieradresse aus.

EXIT (-)
Springt sofort aus dem Wort heraus, in dem es definiert wird. Darf in DO...LOOP bezw. +LOOP, und >R...R> nicht verwendet werden.

F* (f1,f2 - f1*f2)
Multipliziert zwei Gleitkommazahlen und legt das Ergebnis auf dem stack ab.

F+ (f1,f2 - f1+f2)
Addiert zwei Gleitkommazahlen und legt die Summe auf den stack.

F- (f1,f2 - f1-f2)
Subtrahiert zwei Gleitkommazahlen.

F. (f -)
Zeigt eine Gleitkoammazahl an.
1.ØE-4≤f<1.ØE9 wird ohne Exponent und mit Dezimalpunkt an der richtigen Stelle angezeigt. Zahlen außerhalb dieses Bereichs werden in der Standardform f'En, mit Ø≤f'<1Ø und -64≤n≤62 angezeigt.
Die Eingabe ist beliebig, es werden jedoch nur 6 signifikante Ziffern übernommen, weitere Zahlen werden ignoriert. Gleitkommazahlen werden in drei Bytes als binär codierte Dezimalzahl für die Mantissen und Dezimalzahlen für die Exponenten gespeichert.

F/ (f1,f2 - f1/f2)
Dividiert zwei Gleitkommazahlen.

FAST "Schnelle" Arbeitsweise (Fast-Mode), läuft ohne Fehlerprüfung (siehe SLOW).

FIND (- Kompilieradresse)
Legt die Kompilieradresse des ersten Worts im Eingabebereich auf den stack, wenn es im Context-Wörterbuch definiert ist, sonst Ø.

FNEGATE (f - -f)
Negation einer Gleitkommazahl.

FORGET Name (-)
Löscht das Wort "Name" und alle danach definierten Wörter aus dem Wörterbuch.

FORTH Macht das Standard-Wörterbuch FORTH zum Context-Wörterbuch.

HERE (- Adresse)
Legt die Adresse des ersten Bytes nach dem Wörterbuch auf den stack.

HOLD (Zeichen -)
Wird bei formatierter Ausgabe zur Übernahme eines Zeichens in den Pad benutzt.

I
(- Schleifenzähler)
Die oberste Zahl des RETURN-stack wird in den DATA-stack kopiert. Dies ist entweder der Schleifenzähler der innersten DO...LOOP Schleife, oder die zuletzt mit >R in den RETURN-stack übertragene Zahl.

I'
(- Grenzwert)
Kopiert die zweite Zahl aus dem RETURN-stack in den DATA-stack. In einer Do-Schleife ist dies der Grenzwert der Schleife.

CI IF
(n -)
Wird in der Form
IF...THEN
oder
IF...ELSE...THEN
verwendet.
In der ersten Form werden die Wörter zwischen IF und THEN ausgeführt, wenn n ungleich Ø ist, sonst werden sie übersprungen.
In der zweiten Form wird IF...ELSE ausgeführt und ELSE...THEN übersprungen, wenn n ungleich Ø, andernfalls wird IF...ELSE übersprungen und ELSE...THEN ausgeführt.

IMMEDIATE
(-)
Das neueste Wort im Current-Wörterbuch wird, auch im Kompiler-Modus, sofort ausgeführt.

IN
(Anschlußadresse - Datenbyte)
Übernimmt ein Datenbyte von einem I/O-Anschluß.

INKEY
(- ASCII Code)
Fragt die Tastatur ab. Übernimmt den ASCII Code einer gedrückten Taste auf den stack, sonst eine Ø.

INT
(f - n)
Wandelt eine Gleitkommazahl mit Vorzeichen in eine einfache Ganzzahl um.

INVIS
Unterdrückt den Kopiervorgang und OK auf dem Bildschirm (s. VIS)

J
(- Schleifenzähler)
Kopiert die dritte Eintragung im RETURN-stack auf den DATA-stack. Das ist entweder der Schleifenzähler der zweitinnersten DO-Schleife oder die Zahl, die durch das drittletzte >R im RETURN-stack abgelegt wurde.

LEAVE
(-)
Beendet eine DO-Schleife beim nächsten LOOP oder +LOOP, indem es den Schleifenzähler gleich dem Grenzwert setzt.

LINE
Interpretiert den Eingabebereich als normale FORTH-Zeile.

LIST Name
(-)
Listet das Wort "Name" auf dem Bildschirm an. Das Wort muß durch :, DEFINER oder COMPILER definiert sein. Es werden maximal 18 Zeilen angezeigt. Durch Betätigen einer beliebigen Taste wird weitergeschaltet. Mit BREAK kann der Vorgang unterbrochen werden.

CI **LITERAL** (n -)
Kompiliert den TOS als Zeichenfolge in eine Wortdefinition.

LOAD Name (-)
Sucht einen Datensatz "Name" auf der Kassette und lädt ihn in den Speicher im Anschluß an das bereits gespeicherte Wörterbuch. Zeigt die Namen aller gefundenen Datensätze auf dem Bildschirm an. (siehe **SAVE**).

CI **LOOP** (-)
Wie **+LOOP**. Addiert jedoch immer 1 zum Schleifenzähler.

MAX (n1,n2 - max(n1,n2))
Ermittelt die größere von zwei Zahlen.

MIN (n1,n2 - min(n1,n2))
Ermittelt die kleinere von zwei Zahlen.

MOD (n1,n2 - Rest aus n1/n2)
Der Rest hat dasselbe Vorzeichen wie der Dividend n1.

NEGATE (n - -n)

NUMBER Nimmt eine Zahl aus dem Anfang des Eingabebereichs. Legt die Zahl und eine Adresse ≠ Ø auf den stack. Die Adresse ist die Kompilieradresse eines Literal-Kompilers. Mit EXECUTE wird die Zahl als Literal (Zeichenfolge) ins Wörterbuch kompiliert. Für Ganzzahlen ist dies 41Ø2, für Gleitkommazahlen 4181.
Ist keine gültige Zahl vorhanden, wird Ø auf den stack gelegt.

OR (n1,n2 - n1 **OR** n2)
Bitweise Boolesche Operation.

OUT (Datenbyte,Anschlußadresse -)
Überträgt ein Datenbyte an eine I/O-Adresse.

OVER (n1,n2 - n1,n2,n1)

PAD (- 9985)
Legt die Adresse des 254 Byte langen Arbeitsbereichs (Pad) auf dem stack ab.

PICK (n1 - n2)
Kopiert die Eintragung an Stelle n1 des stack an den TOS. n1 selbst geht verloren. ERROR 7, wenn n1 ⩽ Ø.

PLOT (x,y,n -)
Gibt ein Pixel an Stelle x,y mit dem Plotting-Modus n aus.
n = Ø schwarz
1 weiß
2 keine Änderung
3 Änderung
Ist n>3. wird der Vierer-Rest gebildet und als Plotting-Modus verwendet.

QUERY	Löscht den Eingabebereich und nimmt Zeichen an, bis ENTER gedrückt wird. Der Bereich kann aufbereitet werden. Er ist auf 22 Zeilen begrenzt.
QUIT	(-) Löscht den RETURN-stack und den Eingabebereich und übergibt die Steuerung an die Tastatur.
R>	(- Eintragung von RETURN-stack) Überträgt die erste Eintragung aus dem RETURN-stack in den DATA-stack.
REDEFINE Name	(-) Ersetzt das Wort "Name" durch das neueste Wort im Wörterbuch. Überarbeitet das Wörterbuch um Änderungen zu berücksichtigen. Meist verwendet in der Form EDIT Name REDEFINE Name
CI REPEAT	(-) Verwendet in der Folge BEGIN...WHILE...REPEAT. Veranlaßt einen Rücksprung direkt hinter BEGIN.
RETYPE	Gibt dem Benutzer die Möglichkeit, Daten einzugeben. Setzt die MARKE (Cursor) auf ▯ (vgl.QUERY, das vorher den Eingabebereich löscht).
ROLL	(n -) Übernimmt die Eintragung an Stelle n des stack, überträgt sie an den TOS und verschiebt alle Zahlen nach hinten. n geht verloren. ERROR ? wenn n⇐.
ROT	(n1,n2,n3 - n2,n3,n1)
CI RUNS>	siehe COMPILER
SAVE Name	(-) Sichert das komplette Wörterbuch aus dem RAM-Bereich in einen Datensatz "Name" auf Band. Erzeugt ein Geräusch im eingebauten Lautsprecher. Siehe VERIFY und LOAD, ebenso BSAVE, BVERIFY und BLOAD.
SIGN	(n -) Übernimmt ein Minuszeichen zur formatierten Ausgabe in den Pad, wenn n negativ ist.
SLOW	(-) "Langsame" Arbeitsweise mit Fehlerprüfung. Siehe FAST
SPACE	(-) Gibt einen Zwischenraum auf dem Bildschirm aus.
SPACES	(n -) Gibt n Zwischenräume auf dem Bildschirm aus.
SWAP	(n1,n2 - n2,n1)
CI THEN	verwendet mit IF

TYPE (Adresse,n -)
Zeigt n Zeichen aus dem Speicher an, Beginn bei Adresse.

U* (un1,un2 - doppeltlang (un1*un2))
Multipliziert zwei einfachlange Zahlen zu einem doppeltlangen Produkt (ohne Vorzeichen).

U. (un -)
Gibt eine einfachlange Zahl ohne Vorzeichen mit nachfolgendem Zwischenraum auf dem Bildschirm aus.

U/MOD (ud1,un2 - un3,un4)
Alle Rechenvorgänge ohne Vorzeichen. Die doppeltlange Zahl ud1 wird durch die einfachlange Zahl un2 dividiert. Das Ergebnis sind die einfachlangen Zahlen un3, Rest, und un4, Quotient.

U < (un1,un2 - MERKER)
Merker ist 1, wenn un1 un2.

UFLOAT (un - f)
Wandelt einfachlange Zahl ohne Vorzeichen in Gleitkommazahl um.

CI **UNTIL** (n -)
Verwendet in **BEGIN...UNTIL.** Springt zurück, wenn n = Ø.

VARIABLE Name (n -)
Speichert eine Variable "Name" mit dem Anfangswert n.

VERIFY Name (-)
Vergleicht Wörterbuch auf Band gegen Wörterbuch im Speicher. Siehe **SAVE.**

VIS Ermöglicht Kopiervorgang und Anzeige von OK auf dem Bildschirm.

VLIST Listet Wörterbuch auf dem Bildschirm auf.

VOCABULARY Name (-)
Definiert ein neues Wörterbuch mit dem gegebenen Namen.

CI **WHILE** (n -)
Verwendet in **BEGIN...WHILE...REPEAT.** Ist n = Ø verzweigt es hinter **REPEAT.**

WORD Text (Begrenzer - Adresse)
Nimmt einen Text aus dem Eingabebereich bis zum Begrenzer und kopiert ihn in den Pad. Beginnt dort beim zweiten Byte. Stellt die Länge des Textes ohne Begrenzer in das erste Byte des Pad und legt die Adresse des ersten Bytes des Pad in den stack.
Es werden maximal 253 Zeichen aus dem Eingabebereich genommen. Sind dort mehr Zeichen vorhanden, hat das erste Byte des Pad den Wert 254.
Führende Begrenzer werden ignoriert.

XOR Bitweise Boolesche Operation. Exclusiv oder.

I **[** (-)
Eröffnet Interpreter-Modus.

] (-)
Eröffnet Kompiler-Modus.

155

ANHANG D

SCHNELLER ÜBERBLICK FÜR FORTH-KENNER

Die FORTH-Sprache Ihres Jupiter ACE basiert auf der Version FORTH-79. Die grundsätzlichen Unterschiede zur Basisversion sind:

1. Ihr ACE braucht keinen speziellen Bildschirm. Ein- und Ausgabe wird mit Kassettenrecorder durchgeführt, dabei wird entweder ein Wörterbuch in compilierter Form abgespeichert (Liste compilierter Adressen), oder es werden unbearbeitete Bytes aus dem Speicher übernommen. Siehe **SAVE**, **VERIFY**, **LOAD**, **BSAVE**, **BVERIFY**, **BLOAD**.

2. Der ACE kann selbstverständlich Wörter decompilieren - siehe **LIST** und **EDIT**. Er kann auch bereits compilierte Wörter im Wörterbuch ändern und dabei alle Compilierungsadressen und MERKER anpassen - siehe **REDEFINE** und **LOAD**.

3. Es gibt einige Erleichterungen für die Gleitkommaarithmetik - **F+**, **F-**, **F***, **F/**, **F.**, **FNEGATE**, **INT**, **UFLOAT**.

4. **DEFINER...DOES>** ersetzt **:...CREATE...DOES>** um neue definierende Wörter zu definieren; dazu gibt es ein korrespondierendes Paar **COMPILER...RUNS>** für die Definition neuer compilierender Wörter.

5. Es gibt eigene Wörter: **ASCII**, **AT**, **BEEP**, **CALL**, **CLS**, **FAST**, **IN**, **INKEY**, **INVIS**, **LINE**, **NUMBER**, **OUT**, **PLOT**, **RETYPE**, **SLOW**, **VIS**.

6. Folgende Wörter sind im ACE FORTH nicht vorhanden: **'**, **+!**, **-TRAILING**, **79-STANDARD**, **>IN**, **?**, **CMOVE**, **COMPILE**, **COUNT**, **DEPTH**, **EXPECT**, **FILL**, **KEY**, **MOVE**, **NOT**, **STATE**, **[Compile]**.

   Viele dieser Wörter sind im Buch als Beispiele definiert.

REGISTER

!	35,37,62
"	15,1Ø8
#	96
#>	96
#S	96
'	1Ø5
(	25
)	25
*	19,96
*/	21,96
*/MOD	2Ø,96
+	17
+!	37
+LOOP	49
,	1Ø2
-	19
_	25
-TRAILING	85
.	17
."	15
.S	127
/	19,96
/MOD	19,96
:	14
;	14,1Ø3,12Ø
<	41,42,9Ø
<#	96
<BUILDS	1Ø4
=	42
>	42
>R	52
?	37
@	8,16,144
C	144
G	59,144
?DUP	43
@	35,37,62
[	117
]	117
←	31,43
→	31
↑	31
↓	31
~	85
CI	145
Ø	6
Ø<	41,42
Ø=	42
Ø>	42
1	6
1+	2Ø
1-	2Ø
2!	78
2+	2Ø
2-	2Ø
2@	78
2CONSTANT	1Ø5
2DROP	78
2OVER	78
2PICK	78
2ROLL	78
2ROT	78
2SWAP	78
2VARIABLE	1Ø5

A

ABORT	83,126,128,13Ø
ABS	2Ø,98
Absolutwerte	69
Adresse	36
Adreßbus	132
Akkumulator	98
Aktionsteil	1Ø4
ALLOT	1Ø3
AND	92
arrays	1Ø1,1Ø7,11Ø
ASCII	59,62,93
ASCII-Code	59
ASSEMBLER	131
AT	63
Auffüllbereich	84

B

BASE	88,126
BASIC	41,65,82
Basis	54
Bedingung	4Ø,92
BEEP	55
BEGIN	46
Begrenzer	84
Bereiche	1Ø7,11Ø
Bereichsvariable	1Ø1
Bildschirminhalt	74
Binär codierte Dezimalzahl	99
Binärschreibweise	88
Binärsystem	88,89
binary code decimal	99
Bit	89
BLOAD	13,75
Boolesche Operationen	92
Box	36
BSAVE	75
bugs	3Ø
Byte	36
"Bytes"	75

C

C	11
C!	62
C,	1Ø2
C@	62
CALL	129
CAPS LOCK	11
Character	62
CLS	11,63

CMOVE 112
Codefeld 101,105
Codefeld-Adresse 119
COMPILER 117,119
CONSTANT 35
CONTEXT 116,125
Context-Wörterbuch 114
CONVERT 98
COS 80
Cosinus 80
COUNT 85
CPU 129
CR 15
CREATE 101
CURRENT 116,125
Current-Wörterbuch 114

D

D+ 95
D. 97
D.R. 100
D< 96
D0< 98
D0= 98
D->PAD 97
DABS 98
DATA-stack 52,123
Datenbus 132
DECIMAL 88
DEFINER 103,117
DEFINITIONS 114
Definitionsteil 104
DELETE 11,31
DELETE LINE 11
delimiter 84
Dezimalpunkt 77
Dezimalsystem 87
Diagramm 67,72
Diagonalschritt 69
"Dict.:" 13,73,75
DNEGATE 96
DO 48
DOES> 103
DOPPEL 23
Doppellängen-Arithmetik 95
DRAW 68
DROP 24
Dualsystem 87
Dummy 45
DUP 24

EAR 12
EDIT 31,34
EINGABE 82
Eingabebereich 81,82
Elemente 110
ELSE 40
EMIT 59
Endwert 49
ENTER 8,31

E

Entscheidungen 39,40
ERROR 10,14,16,142
EXECUTE 84,119
EXIT 51
Exponent 54
Exponentialschreibweise 77
Exponenten-Byte 99

F

F* 77
F+ 19,77
F- 77
F. 19,77
F/ 77
"falsch" 41
Fakultät 44
FAST 131
Fehleranzeigen 33,142,143
Fehlerprüfungen 131
FILL 113
FIND 83
flag 41
Floating point 77
Flußdiagramm 39
FNEGATE 77
FORGET 33
Formatierung 96
FORTH 114
Forth-79 155

G

Ganzzahlarithmetik 28
Gleitkommazahlen 77,99
GR 61
GRAPHICS 11
Graphic-Modus 59
Grenzwert 50

H

Halbfettdruck 9
Halbtonschritte 55
halt 130
Header 101
HERE 121
HEX 91
Hex 88
Hexadezimalzahlen 87
Hilfsregister 130
HOLD 96

I

I 49
I' 50
IF 40
IMMEDIATE 117
IN 132
Index 110
INKEY 81
INT 77

I

INTEGER 83
integer 77
Interpreter-Modus 117
interrupt 13Ø
INVERSE VIDEO 11,59
Inverse Video-Modus 59
INVIS 67
I/O Adressen 132

J

J 5Ø
J/N 81

K

Kassette 12,73
Kilo Byte 91
Kommentar 25,33,118
Kompiler-Modus 117
Kompiler-Wörter 119
Kompilieren 119
Kompilier-Adresse 119
Kompiliertes Wort 12Ø
Kompilierzeit-Wirkung 118
Konstantenbereich 112

L

Längenfeld 1Ø5
Laufzeit-Wirkung 118
Lautstärkeeinstellung 74
LEAVE 51
Leitwort 114,124
LINE 82
LIST 3Ø,34
Listing 33,118
LITERAL 118
LOAD 12,13,74
Logarithmus 57
LOOP 48

M

MARKE 8,1Ø,31
Maschinencode 129
Maschinen-stack 13Ø
Masken 94
MAX 2Ø
mehrdimensionale Bereiche 112
memory-mapped 135
MERKER 41
MIC 12
MIN 2Ø
mnemotechnische Form 129
MOD 19
MOVE 112

N

Name 1Ø1
Namensfeld 1Ø5
Namenslängenfeld 1Ø5
NEGATE 2Ø
negativer Step 49
negative Zahlen 89
NUMBER 83

O

OK 8,13,14
Operanden 23,24
Operandenfeld 12Ø
OR 92
OUT 132
OVER 26

P

PAD 84,97
Pad 84
Parameter-Feld 1Ø1
Peripherie 132
PICK 26
Pixel 67
PLOT 67
PLOT-Status 7Ø
Plotting-Modus 67
Port 132
Port-Adressen 135
positive Zahlen 89
Potenz 37,54
PRIM 53
PRIM? 53
PRIMES 53
Primzahl 53
Probier-Divisor 53
Prüfwort 42
Pseudozufallszahl 7Ø

Q

QUERY 82
QUIT 82
Quotient 19

R

R 52
RAM 36,123
RAND 71
Raster 6Ø
REDEFINE 31
Register 13Ø
Reihenzähler 5Ø
Reihenzählschleife 5Ø
Rekursion 44
relative Adresse 1Ø2
REPEAT 47
Rest 19
Restart-Befehle 13Ø
return adress 52
RETURN-stack 52,123

159

R

RETYPE 82
RND 71
ROLL 27
ROM 36,123,129
ROT 26
Rücksprung-Adresse 52
RUNS> 12Ø

S

S->D 1ØØ
SAVE 74
Schaltkreise 134,135
Schleife 47,48,5Ø
Schleifenzähler 51
Schutzstellung 1ØØ
SHIFT 1Ø
SIGN 97
SIN 79
Sinclair ZX81-Anschlüsse 136
Sinus 79
SLICE 1Ø8
Slicing 1Ø8
SLOW 131
Sofortwort 1Ø5,117
SPACE 1Ø,63
SPACES 63
Speichererweiterung 124
Speicherplätze 36
Spitze des stack 39
SQ 24,34
stack 17
stack underflow 22
Stammwörter 131
Stapel 17
step 49
Steuerzeichen 59
Strings 1Ø7
Stringvergleich 1Ø8
Substrings 1Ø8
SWAP 24
SYMBOL SHIFT 8
Systemvariable 69,88,123,124

T

TAB 65
TAN 8Ø
Tangens 8Ø
THEN 4Ø
Tondauer 55
Tonhöhe 55
Tonlänge 55
top of stack 39
TOS 39
TYPE 62,84

U

U* 96
U. 89
U/MOD 96
U< 9Ø
UFLOAT 78
Umwandlungsadresse 83
Underflow 124
Unterlauf 124
UNTIL 47

V

VARIABLE 35
"Vater"-Wörterbuch 114
Verbindungsadresse 115
VERIFY 73,74
Verkettung 1Ø1
Verkettungsfeld 1Ø5
Video RAM 123
VIS 67
VLIST 9,14,115
VOCABULARY 114
Vorsatz 1Ø1

W

"wahr" 41
WHILE 47
WORD 83
Wortbegrenzung 84
Wörterbuch-Definition 114
Wörterbuch-Verkettung 116

X

X-Koordinate 67
XOR 92

Y

Y-Koordinate 67

Z

Z8Ø-Maschinencode 129
Zahlenbasis 88
Zahlensysteme 87
Zehnersystem 87
Zeichenketten 1Ø7
Zeichensatz 59,137
Zufallsgenerator 7Ø
Zweidimensionale Bereiche 11Ø
Zweierkomplement 95
Zweierkomplement-Methode 89

Gescannt von Dirk Brühl.
Bereinigt und OCR von Michael Kalus.
(Adobe Acrobat Pro DC)
Mai 2020

Der Autor

Steven Vickers *studierte Mathematik, zunächst am King's Kollege Cambridge und später an der Universität von Leeds, wo er zum Doktor der Mathematik promovierte. Danach wandte sich Vickers der Computer-Programmierung zu und hatte wesentlichen Anteil an der Entwicklung des Sinclair ZX81 und ZX Spectrum.*

Er ist auch Autor des meistverbreiteten Computer-Handbuchs für den ZX81 und des Handbuchs für den ZX Spectrum.

1982 gründete er zusammen mit Richard Altwasser die Firma Jupiter Cantab Ltd. und entwickelte den Jupiter ACE.

Die Übersetzer

Friedrich Haas *ist ein 'Oldtimer' unter den Computer- und Programmier-Spezialisten. Schon als junger Ingenieur arbeitete er in den sechziger Jahren mit an der Entwicklung des ersten deutschen Großrechners, des ER 56 der Firma Standard Elektrik Lorenz AG.*

Zusammen mit ***Wolfgang Diez*** *entwickelte er bereits 1960 das erste Fertigungsplanungs- und Steuerungsprogramm der Welt für eine mehrstufige Parallelfertigung mit festen Auftragsgrößen auf Mehrfunktionsmaschinen.*

Beide sind, nach langjähriger Praxis in Industrie und Handel, während der sie auch viele Projekte gemeinsam verwirklichten, heute als freie Unternehmensberater tätig.

Die Programmiersprache FORTH ist eine Sprache, die sich im Bereich der Mikrocomputer immer mehr durchsetzt. Für viele Anwendungen ist sie hervorragend geeignet. Mit FORTH kann man, einmal damit vertraut, schnelle und effektive Programme bei hoher Speicherplatzausnutzung schreiben.

Dieses Handbuch, welches für den Benutzer des Jupiter ACE geschrieben wurde und jedem Käufer des ACE zur Verfügung gestellt wird, kann auch unabhängig davon eingesetzt werden, um die hochinteressante Sprache FORTH in klarer und gut lesbarer Darstellung kennenzulernen.

Herausgeber

reflecta electronic GmbH
Berlichingenstr. 9
8540 Schwabach

die auch das Alleinvertriebsrecht für den Jupiter ACE
in der Bundesrepublik Deutschland besitzt